

# تحليل وتصميم الخوارزميات

## Algorithms Design and Analysis

تأليف

الدكتور	الأستاذ المساعد	الباحث
حسن ياسين طعنه	هند رستم محمد شعبان	حسن ثابت رشيد كرماشة

*hind\_restem@yahoo.com*  
*hassan\_thabit@yahoo.com*

تحليل وتصميم الخوارزميات  
Algorithms Design and Analysis

الفهرس .....

الفصل الأول: مقدمة (Introduction)

- 1-1: مقدمة في الخوارزميات ( Algorithms Introduction )
- 2-1: كيفية تحليل الخوارزمية (Algorithm Analysis)
- 3-1: الوقت الكلي لتنفيذ الخوارزمية ( Execution Time )
- 4-1: الحالات الأفضل والأسوأ والمتوسطة للتحليل (Best & Worst & Average Cases Analysis)
- 5-1: الصيغ التقريبية (Asymptotic notation)
- 6-1: الصيغ الشائعة لأوقات التنفيذ ( The Times of Executive Notation )
- 7-1: الاستدعاء الذاتي لشجرة التسيطات أو الاستدعاءات ( Recursion ) (Tree)
- 8-1: قياس الانجازية (Performance Measurement)

الفصل الثاني: الترتيب (Sorting)

- 1-2: خوارزميات الترتيب (Sorting Algorithms)
- 2-2: أنواع الترتيب (Types of Sorting)
- 3-2: خوارزميات الترتيب الداخلي ( Internal Sort Algorithms )
  - 1- ترتيب الاختيار ( Selection Sort )
  - 2- ترتيب الفقاعي (Bubble Sort)
  - 3- ترتيب الإضافة (Insertion Sort)
  - 4- ترتيب شيل (Shell Sort)
  - 5- الترتيب السريع ( Quick Sort )
  - 6- ترتيب الأساس (Radix Sort)
  - 7- ترتيب المؤشرات (Pointers Sort)
  - 8- ترتيب الشجري لشجرة البحث الثنائية ( Tree Sort )
  - 9- Topological sorting
- 4-2: خوارزميات الترتيب الخارجي (External Sort Algorithms)
  - 1- ترتيب الدمج (merge Sort)
  - 2- ترتيب الدمج المتوازن ذو الممرتين (Balanced Two-Way Merge Sort)

### الفصل الثالث: البحث (Searching)

- 1-3: البحث (Searching)
- 2-3: البحث التسلسلي (Sequential Search)
- 3-3: البحث الثنائي ( Binary Search Algorithm)
- 4-3: البحث في الشجرة الثنائية (Binary Search tree)
- 5-3: تعقيد خوارزمية البحث (Algorithm search Complexity)

### الفصل الرابع: الأمثلية في مسائل تصميم الخوارزميات (Optimization in Algorithms Design Equations)

- 1-4: المخططات (Graphs)
- 2-4: أنواع المخططات (Type of Graphs)
- 3-4: طول المسار (Path Length)
- 4-4: طريقة الجشع أو الطماع (Greedy Method)
- 5-4: مسألة الجراب ( Knapsack Problem)
- 6-4: استخدام قاعدة الطماع في إيجاد أمثلية البيانات

### الفصل الخامس: البرمجة الديناميكية (Dynamic Programming)

- 1-5: البرمجة الديناميكية (Dynamic programming)
- 2-5: أمثلة على البرمجة الديناميكية
- 3-5: تجميع البيانات (Data clustering)
- 4-5: خوارزمية (Dijkstra)
- 5-5: أمثلة لتطبيق خوارزمية (Dijkstra)
- 6-5: المخططات المتعددة المراحل (Multistage graph)
- 1-6-5: الطريقة التصاعدية (Forward approach)
- 2-6-5: الطريقة التناقصية (Backward approach)
- 3-6-5: طريقة اقتفاء الأثر رجوعاً (Back Tracking)

# الفصل الأول

## مقدمة

### (Introduction)

## 1-1: مقدمة في الخوارزميات (Algorithms Introduction) :

- الخوارزمية هي مجموعة محددة من التعليمات (خطوات الحل) التي تؤدي إلى إنجاز وظيفة (مهمة) معينة ويجب أن تتوافق فيها الشروط التالية :
1. المدخلات (Input) : صفر أو أكثر من القيم .
  2. المخرجات (Output) : قيمة واحدة على الأقل .
  3. الوضوح (Definiteness) : كل خطوة فيها (الخوارزمية) واضحة المعاني وغير غامضة أي يجب أن نفهم من قبل جميع الناس (مفهوم الحسابات).
  - و على سبيل المثال نأخذ العبارة "Add 6 or 7 to X" هذه العبارة غير مفهومة بها في الخوارزمية لأنها عبارة غير واضحة .
  4. المحدودية (Finiteness) : كل خطوات الخوارزمية يمكن حلها في فترة زمنية محددة ، والتوضيح تلك نأخذ العبارة " قسم الرقم (10) على (3) بدقة عالية (كاملة) " هذه العبارة غير محدودة ويجب أن لا يسمح بها دخال البرنامج .
  5. الفعالية (Effectiveness) : كل خطوة تكون ممكنة الحل أو التنفيذية ، مثال تلك العبارة "  $0/3$  " لا يمكن حلها أبداً .

يمكن لنا أن نوضح الفرق بين الخوارزمية والبرنامج حيث أنه في النظرية الاحتمالية يوجد فرق بين الخوارزمية والبرنامج ، ففي الخوارزمية يجب أن تتوافق الشروط الخمسة الأتفة للذكر ويمكن وصفها بطرق عديدة مثل لغة طبيعية مع التأكيد على شرط الوضوح ، لغة خوارزمية (Pseudo-code) ، مخططات انسيابية (Flow chart) ، بينما يمكن في البرنامج عدم تحقق الشرط الرابع حيث إن نظام التشغيل هنا هو الذي يعتمد على البرنامج ويوصف البرنامج بلغة الحاسبة حيث أنه يصمم ليتحكم في تنفيذ مجموعة من الأعمال (Jobs) بحيث عند عدم توفر عمل معين فإنه لا ينتهي من أعماله بل يستقر ويدخل في حالة لتتظار لحين إدخال عمل جديد .

إن لكل لغة برمجية يوجد مترجم أو مفسر ولا يمكن تواجدهما معاً حيث إن المفسر يقوم بتنفيذ البرنامج خطوة خطوة (step by step) بينما المترجم فإنه يتخذ البرنامج كاملاً ويظهر النتائج والأخطاء ، هذا يعني إن البرنامج هو عبارة عن خوارزمية وهيكل يأتى أي أنه طريقة لتنظيم البيانات.

خطوات تطوير البرنامج :

تتم عملية تطوير البرنامج بخمس خطوات رئيسية هي :

1. توصيف المتطلبات (Requirement specification):

هو تحديد المدخلات والمخرجات.

2. التصميم (Design):

هو تحديد العمليات الرئيسية التي تطبق على كل بيان بياني والفقرات وجرد أجهزة معالجة لتنفيذ هذه العمليات.

3. التحليل (Analysis):

هو المفصلة بين الخوارزميات المعروفة التي تحل نفس المسألة تبعاً لمقاييس المفصلة متقناً عليها (تقنيات الوقت، تقنيات الفراغ، التخزين) ، باختيار أفضلها .

4. التحسين والتشفير (Refinement & Coding):

في هذه الخطوة يتم تحديد التمثيل النهائي لكل بيان ثم كتابة الإجراءات لكل عملية على تلك الكيفيات وتكون نسخة متكاملة للبرنامج.

ملاحظة // التحليل يصلح الأخطاء اعتماداً على تقنيات التخزين والوقت بينما التحسين يصلح الأخطاء اعتماداً على النتائج الظاهرة في نهاية البرنامج.

5. التحقق من الصلابة (Verification):

تضمن هذه الخطوة ثلاث جوانب هي :

أ- البرهنة على الصحة (Proving) :

قبل استخدام البرنامج يجب إثبات أنه صحيح حيث يتم استخدام الطرق المعروفة للبرهنة على الصحة.

ب- الاختبار (Testing):

هي عملية توليد نماذج بيانية يعمل عليها البرنامج حيث إن الهدف منها هو إعطاء إشارة على وجود أخطاء في البرنامج.

ج- تشخيص الأخطاء (Debugging):

عملية تحديد مواقع الأخطاء البرمجية في البرنامج وتصحيحها .

ملاحظة : إن التعريف يختلف عن الصلابة فالترقيف هو معرفة شيء قد يكون صحيح أو خطأ بينما الصلابة هي معرفة شيء يجب أن يكون صحيح .

أما النموذج فهو تحقيق تمثيل بياني بالشكل الصحيح.

في علم الحاسبات يتم أولاً الاختبار وبعدها يتم البرهنة بينما في علم الرياضيات يتم البرهان وبعده الاختبار.

## 2-1: كيفية تحليل الخوارزمية (Algorithm Analysis)

تحليل الخوارزمية هو تحديد الكفاءة للخوارزمية ومن ثم تصنيفها حيث يوجد مقياسين مرتبطين مباشرة بشجرة الخوارزمية هما :

1 - مقياس تعقيدات الفراغ أو التخزين (Space Complexity): هي كمية الذاكرة التي يتطلبها تشغيل البرنامج حتى اكتماله بحيث يعتمد هذا النوع على جزئين :

أ- جزء ثابت : هو مستقل عن خصائص المدخلات والمخرجات حيث يتضمن هذا الجزء فراغ التعليمات (Code space) ، الفراغ المخصص للمتغيرات (Data Space) سواء كانت البسيطة أو المتغيرات المركبة ذات الحجم الثابت إضافة إلى فراغ التراكيب ، ..... الخ .  
 ب- جزء متغير: يتألف من الفراغ الذي يتطلبه البرنامج بالمتغيرات المركبة والتي يعتمد حجمها على مثال المسألة المراد حلها ، إضافة إلى فراغ المكدس المستخدم في التداخل (Reaction).



شكل (1): تحليل الخوارزمية

إن التخزين التبادلي يمكن توضيحه بالمتغيرات التي يدخلها البرنامج (أي يدخل قبتها) وكذلك يتحكم بأسمائها فهي متعدة على إدخال البرنامج.

أما القيم المركبة فهي المسفوفة التي تمثل بالمكدس حيث المؤشر هو (Sp) عمادياً وبرمجياً يسمى (Top) ، وفيما يخص تصنيف الخوارزميات فإن المهم هنا هو مستوى المسفوفة أي المكدس.

يمكن صياغة تعقيدات الخزن للبرنامج كالآتي :

ثابت

Code segment
Data segment
Heap segment
Stack segment

إن جزء (Code Segment) يمكن تمثيله كما الحوال الجاهزة ، أما (Heap Segment) فيكون المتغيرات التي يستخدمها البرنامج .

وعليه يمكن صياغة تعقيدات الفراغ  $S(p)$  للبرنامج (p)

$$S(p) = \text{Const} + Sp$$

حيث إن :

Const : تمثل جزء (Code segment) والمتغيرات البسيطة .

Sp : تمثل خصائص المثال .

2- تعقيدات الوقت (Time Complexity) : هي كمية الوقت التي يتطلبها تنفيذ البرنامج حتى اكتماله ويتألف من :

$$T(p) = \text{Const} + tp$$

حيث :

Const : يمثل ثابت خاص بوقت الترجمة أو التفسير .

Tp : يمثل وقت تشغيل البرنامج .

مثال 1 // لبيان تعقيدات الفراغ (الخزن) والوقت لدالة معينة (بلغة C++) :

```
Float abc(float a,float b,float c)
{return(a+b+5*c+(a+b+c)/(a+b)+4.0); }
```

تعقيدات الفراغ أو الخزن :

تطلب الدالة (abc) خمسة خلايا خزن لثمة قيم المتغيرات (a,b,c) والمتغير الذي يحمل اسم الدالة وعنوان العونة (Return address) وهو خزن ثابت لا يعتمد على خصائص المثال (a,b,c).

$$S_{abc}(a,b,c) = 0$$

إن قيمة الصفر هنا تعني إن الخزن ثابت أي لا يعتمد على خصائص المثال .

تعقيدات الوقت : تستخدم صيغة عد الخطوات (Steps Count) لقياس تقدير الوقت حيث إن عدد الخطوات لهذه الدالة يساوي واحد ، ولها فإن :

$$T_{abc}(a,b,c) = 0$$

إن قيمة الصفر هنا أيضا تعني إن الوقت ثابت .



مثال //2 اكتب خوارزمية لإيجاد القاسم المشترك الأعظم (Greatest Common Divisor) لعددين صحيحين .

//الحل

```

Step 0 : [ check m and n ]
    If m <= 0 or n <= 0 then
        Print error.
Step 1: [ test m and n ]
    If m < n then
        Inter change m by n.
Step 2: [find the remainder]
    Divid m by n and let r is
    Remainder we will have 0 <= r < n.
Step 3: [is r = zero]
    If r = 0 then
        GCD = n and exit.
Step 4:[ inter change ]
    M ← n, n ← r go to step 2

```

مثال تطبيق //1

m	n	r
10	6	4
6	4	2
4	2	0

GCD = 2

مثال تطبيق // 2

m	n	r
20	130	
130	20	10
20	10	0

GCD =10

عدد مرات تنفيذ العبارة (Frequency Count) :

إن عدد مرات تنفيذ العبارة يختلف حسب عينة البيانات . حيث يوجد لدينا ما يسمى بوقت التنفيذ المفرد للعبارة (execution time for single).

Total execution time = frequency count \* execution time for single

إن الوقت الكلي لتنفيذ يعتمد على العوامل التالية :

1. نوع الحاسبة (Computer type).
2. لغة البرمجة (Programming language).
3. الوقت التنفيذي الخاص لكل عبارة (Total execution time).
4. نوع المترجم أو المفسر (Compiler and interpreter).

مثال 3// افترض وجود الأجزاء البرمجية الآتية مرقبة من 1 إلى 3 كالآتي :

مثال 1	$x = x + 1;$	$F_c = 1$
مثال 2	$\text{For}(\text{int } i=1; i \leq n; i++)$ $x = x + 1;$	$F_c = n$
مثال 3	$\text{For}(\text{int } i=1; i \leq n; i++)$ $\text{For}(\text{int } j=1; j \leq n; j++)$ $x = x + 1;$	$F_c = n^2$

لو افترضنا أن (  $n = 10$  ) فإن مثال رقم (2) فيه عدد مرات تكرار الخطوة التنفيذية هو (10) وعدد المرات في المثال رقم (3) هو (100).  
 نستنتج من ذلك أن المثال رقم (1) ينفذ أسرع من المثالين (2) و (3) ومثال (2) أسرع من مثال (3).

### 3-1: الوقت الكلي لتنفيذ الخوارزمية (Execution Time):

تنظيم الترتيب للخوارزمية (Order of magnitude of Algorithm) :  
 هو مجموع تكرارات جميع العبارات التنفيذية التي يسويها يحدد التقدير المسبق لوقت تنفيذ الخوارزمية.  
 إن العبارة الغير تنفيذية تعني العبارة التي ليس لها تأثير على البرنامج مثل عبارة التعليق في أي لغة برمجية يمكن استخدامها.

مثال// لدينا مصفوفة  $A$  بحجم  $n$  ، نحسب مجموع كل صف ونحزن قيمته في مصفوفة اسمها  $S$  ، ثم نحسب المجموع الكلي لعناصر المصفوفة  $S$  ، ثم اعطي عدد تكرارات العبارات .

$$\text{Sum} = \sum_{j=1}^n a_{ij}$$

الحل // توجد طريقتين:

```

1 Grandtotal = 0;
For(int k = 1; k <= n; k++)
{ s[k] = 0;
For(int j = 1; j <= n; j++)
{ s[k] = s[k] + a[k,j];
Grandtotal = Grandtotal + s[k];
}
}

```

نلاحظ ان عدد التكرارات يساوي  $2n * n$

```

2- Grandtotal = 0;
For(int k = 1; k <= n; k++)
{ s[k] = 0;
For(int j = 1; j <= n; j++)
{ s[k] = s[k] + a[k,j]; }
Grandtotal = Grandtotal + s[k];
}

```

وهنا نلاحظ انها تساوي  $n^2 + n$

هناك ٢ طرق لحساب خوارزمية CN وخوارزمية CN<sup>2</sup> علماً ان C هي ثابت وقم بعمل مقارنة بين الخوارزميتين من حيث وقت التنفيذ علماً ان:

C الأولى = 10 ، والثانية = 0.5

و  $n = \{ 1, 5, 10, 15, 20, 25, 30 \}$

n	CN	CN <sup>2</sup>
1	10	0.5
5	50	12.5
10	100	50
15	150	112.5
20	200	200
25	250	312.5
30	300	450

ملحظة // اذا كانت قيمة n أقل أو تساوي 20 فان وقت الخوارزمية الثانية أقل من وقت الخوارزمية الأولى أما عند عمليات التكرار أو التحصيلات اقل، لكن بعد هذه النقطة (20) أي (25, 30) فان وقت الخوارزمية الأولى يكون أقل، هذه المرة (يسج العكس)

هدف اليوم بالحصيل وقت التنفيذ للخوارزمية فطبي هناك إنا نجد  $(O(g(n)))$  وهذا يعني ان وقت تنفيذ دنا يستغرق أكثر من  $(C * g(n))$  حيث ان (C) هو كمية ثابتة والـ n تمثل عدد العمليات المطلوبة لمحوط الخوارزمية لتصل لحصلها .

مثلاً:

- 1  $O(1)$  معنى ذلك أن وقت الحساب ثابت مثل  $(x \rightarrow x+1)$  صغر البرنامج
- 2  $O(n)$  وهي أن وقت الحساب ذو طبيعة خطية مثل طباعة جميع عناصر مصفوفة حجمها  $n$  أو مثلاً إيجاد عنصر في قائمة موصولة
- 3 Quadratic (وقت تربيع  $O(n^2)$ ) مثل وقت ترتيب عناصر قائمة باستخدام عناصر قسمة صفائي .
- 4  $O(n^3)$  في الوقت الترتيب لحمل عناصر مصفوفة  $(n \times n \times n = 0)$ .
- 5  $O(2^n)$  يعني أن الوقت الترتيب لحمل جميع عناصر مصفوفة مثلاً مساريًا للعنصر هو استخدام الصيغة الأسية .
- 6  $O(\log(n))$  في الصيغة مثل وقت البحث باستخدام صيغة التوابع ثنائية مثل تلك للوصول إلى عقدة معينة في شجرة ثنائية.

ملاحظة: قيمة  $\log$  دائماً تكون مقصورة بين  $(0)$  و  $(1)$  أي أقسام عشرية .

مثال // هناك اقيم لتاليه  $n = \{1, 2, 4, 8, 16\}$   
 مفرد بتضمينها على نحو زمني الآليه وفازتها بجدول تعرض توضيحها .

$n$	$\log_2 n$	$N \log_2 n$	$N^2$	$N^3$	$2^n$
1	0	0	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	1096	65536

$$\log_2(x) \quad \log_{10}(x) \times 3.322$$

مربعين // الزمن التوال الساعه ووضح عصفه الفرق بين الخوارزميات

- ملاحظة 1/ وقت  $O(n)$  ووقت  $O(n \log n)$  يزيد كل منهما بصورة خطية من التوال الأخرى .
- ملاحظة 2/ عندما يكون حجم تسلسل كثير يصبح الخوارزمية لها بعد وقت كثير مثلاً الخوارزمية التي عدد عناصرها  $O(n \log n)$  تكون على القيمة أو غير عصفه .
- ملاحظة 3/ لمر زمنية التي وقت تنفيذ بالصيغة الأسية فإنه يمكن اعتمادها فقط إذا كانت قيمة  $(n)$  صغيرة، حيث أن كانت قيمة  $n$  صغيرة فإن عدد الحساب قبل والتالي في وقت التعداد أسرع وبالتالي لاغير سيكون واضح .

مثال // هناك لساله الآليه

$$F = a + b^2 + c + (a + b) \cdot (a + b) + 4 \cdot 0$$

عفاً إلى التوال  $(a=2.0, b=3.0, c=0.0)$  .  
 المطلوب / تحديد تعقيدات الوقت وتحديد الحرر عفاً إلى  $(a, b, c)$  في قيم حقيقيه

ملاحظة: عند التعويض بالمتغير  $\frac{2}{3} = 2.33$  ، الخزن يعطي 2.33 في المتغيرات الموجودة في

المرآة \*

مصفوفات الحزم - يوجد لدينا خمسة حاويات حزمة هي a, b, c وعدادات الحزم F وتعتبر الاسم للبيانات

، معنى تلك إن  $C(a,b) = S$  أي لا يوجد جزء منفرد في أي الحاوية لك

في حالة إن تكون المصفوفات الخاصة هذه الدالة n من العزب ما هي مصفوفات الخزن هنا معطى في

n=3

و (a,b,c) قيمها كالتالي.

a	b	c
3	2	1
2	3	2
1	4	1

للحل نقوم بالتالي -

نقوم بعمل جدول كالآتي - معنى ذلك انه يوجد قيم صغيرة بالمصفوفات فالنتيجة لا يساوي صفر ونعتمد على قيم المصفوفات

مصفوفات قوف

في الحالة الأولى وفي الدالة يساوي واحد (Count=1) لأنها تغد مرة واحدة وفي الحالة الثانية القوف يعتمد على عدد القوف (Count=n)

مثال: إيجاد مجموع عناصر مصفوفة بحدة الحد كفي في مصفوفة الدالة

$$Sum = \sum_{i=1}^n ai$$

```

Float Sum(float a[],int n)
{ float S=0.0; //عدد المصفوفات
For(int i=1;i<=n;i++) //عدد المصفوفات {n+1}
S+=a[i]; //n
Return S; //1
}

```

يقسم مثال المسألة بالمغير (n)

مصفوفات الحزم - نطلب الدالة (Sum) متعة حاليًا حزمة لآخرين قيم (I, S, n) وعدادات المصفوفة

[ a والمغير الثاني نعمل اسم الدالة لمتابعتها في عنوان الذاكرة.

وهو خزن ثابت لا يعتمد على حصة الذاكرة (أي لا يعتمد على تغير قيمه n)

$$S_{sum}(n)=0$$

تعريف الوقت

$$T_{sum}(n)=2n+3$$

لاحظ إن العلاقة التي تربط الوقت بعدد العناصر هي علاقة خطية وهي أفضل من التربيعية  
مثلاً، سوف نجد نفس العدد السابق ولكن هذه المرة بطريقة الاستدعاء الذاتي (Recursion).

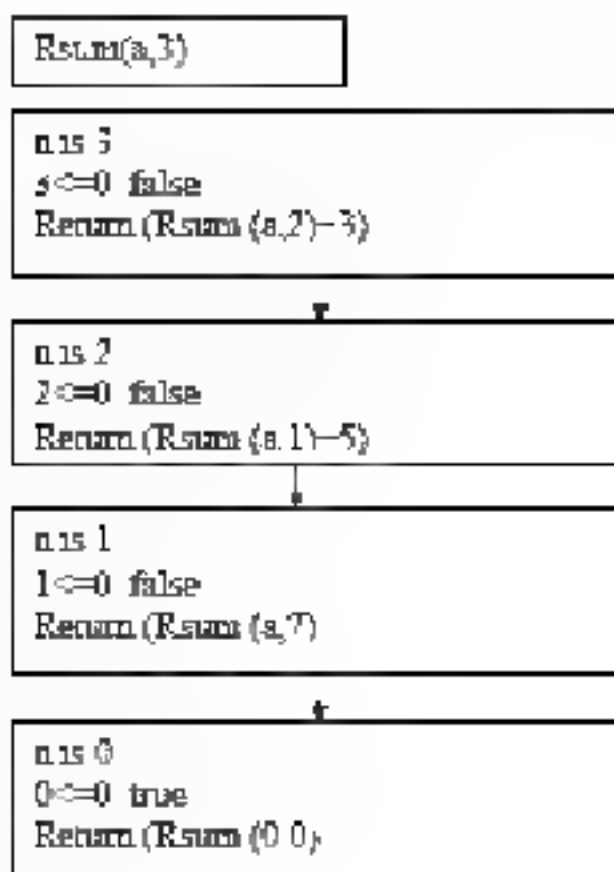
$$Sum = \sum_{i=1}^n a_i$$

$$Rsum(a, n) = \begin{cases} 0.0 & \text{if } n \leq 0 \\ Rsum(a, n-1) + a[n] & \text{if } (n > 0) \end{cases}$$

إذا بالأسطة الصغيرة نرى خاصية أما بالأسطة الصغيرة للفرمجة فهي .

```
Float Rsum(float a[] ,int n)
{if (n<=0) return (0.0);
 Else return (Rsum(a,n-1)+a[n]);
}
```

ونفترض إن المصفوفة {a[1..3]={3 5 7} هي إلى (n=3)



بعضيات القر ع

نضع في فراغ مكان الدخول المتعاقبات الشكلية والمتخلفات العددية وندخل للعدد  
كل تنسيق (استدعاء) يتطلب أربع حالات حرجية (حيزه بقيه (D) وحيزه للفوضر إلى العصفرة  
(E) وحيزه للتعديل الذي يحفل اسم نداله بالإصافة إلى حيزه بطول العدد )  
وهو في معنى السهل (عدد الاستدعاء) ، نصب كالتالي.

معنى السهل (الاستدعاء) = الحجم الأولي في بول السهل - الحجم النهائي في آخر سهل  
1 + |

معنى السهل (الاستدعاء) = 4 = 1 + | 0 n

حدث في الحجم الأولي في أول سهل بعدد به عدد العصفرة 0 ثم نأخذ السهل في الحجم  
النهائي في آخر سهل وهكذا إن يكون (D) أو E فيه أخرى.

$$T_{max}(n) = 4(n-1)$$

من المعقولة أن نلاحظ في الأخير حالة خطية بعدد على قعدة (D) حدث إذا كانت قعدة  
(E) وإذا العرف أن إن قعدة لا (E) كل للحرج

بعضيات أوقات مرفوعة بمتغير متعلق السهل (Recurrence relations) بحسبها  
كالتالي

$$f_{max}(n) = \begin{cases} 2 & \text{if } n \leq 0 \\ 2 + f_{max}(n-1) & \text{if } n > 0 \end{cases}$$

إن تبقي (2) نأخذ الفرق المطلوب لاستدعاء كل تنفيذ ، إن (f<sub>max</sub>(n-1)) فهي نأخذ وقت  
الذي يكون آخر استدعاء  
إن خطوة (Else) بعدد خطوة متعلقة بذلك فهي نأخذ القعدة (D) وهي الخطوة التي  
تحويلاً تبقي (1)  
ولكن هذه العلاقة الداخلية تستخدم طريقة من طرق الحل وهي طريقة التكراري  
(Iterative Substitution) كالتالي

$$\begin{aligned} f_{max}(n) &= 2 + f_{max}(n-1) \\ &= 2 + 2 + f_{max}(n-2) \\ &= 2(2) + f_{max}(n-2) \\ &= 2(2) + 2 + f_{max}(n-3) \\ &= 3(2) + f_{max}(n-3) \\ &= m(2) + f_{max}(n-m) \end{aligned}$$

و عندما  $(m=n)$  تأتي

$$\begin{aligned} & 2n + t_{\text{main}}(n + n) \quad , \quad n > 0 \\ & = 2n + t_{\text{main}}(0) \\ & = 2n + 2 \end{aligned}$$

بعض المتغيرات في تكون ثابت  $(n-n)$  فقد تكون ثابت  $(n-1)$  و أي قيمة تحتوي تحت في  
القيم في تكون ثابت  $(n-m)$ .

مثال : إيجاد مجموع عنصرين متتاليين كما في المصفوفة  $A$  التالية :

$$C_{m+n} = A_{m+n} + B_{m+n}$$

```
Void Add(type a[][size], type b[][size], type c[ ][size] in m, m+n, n)
{ for(int i=1; i<=m; i++) ... m+1
  For (int j=1; j<=n; j++) ... n
    C[i,j]=a[i,j]+b[i,j]
}
```

نلاحظ في مثال المسألة يتصلب بالمتغيرات  $(m, n)$  حيث :

تتصلب المتغيرات : متطلب أقله (الحد الأدنى) قبل ذلك حزمة مغرر قيم المتغيرات  $(m, n, b, a)$   
بالإضافة إلى عنوان الذاكرة ونلاحظ أن الخوار لا يتوقف على حاصلين المتغيرات في

$$S_{\text{main}}(m, n) = 0$$

في العادة إلى تحديد الوقت فهي كالآتي

$$\begin{aligned} & m + 1 \\ & 5m \\ & (m + 1 + m)m \\ & (2m + 1)m \\ & 2mm + m + m + 1 \\ & T_{\text{main}}(m, m) = 2mm + 2m \end{aligned}$$

في هذه العلاقة تكون متصلة في حصة  $(m > n)$  أما عندما تكون  $(m = n)$  فإنه يتصلب الحد  
تسمى لا  $(For)$  في ذلك حفظ تعقيد الذاكرة لتصبح :

$$T_{\text{main}}(m, m) = 2mm + 2m + 1$$



مثلاً: اوجد تعقيد الزمن والذاكرة لسلسلة عداد فيبوناتشي (Fibonacci) التي هي متتابعة من الأعداد حيث كل عدد

أول خمس قهـ (0,3) وهما انداز قهر المستلحة حيث أن عطلة انحصارهم على القيمة التالية من خلال جمع العنصر المتعلقين كالتالي  
0.1, 1, 2, 3, 8, 13.

هذا" ينظر حد حد من الحصول عليه من خلال جمع لعنصر السبق في هذا كل (٢٠)  
فصل الحد الذي في القتال في يوم يومه

$$FQ = 0$$

$\beta = 1$

$$f_{\pi} = \sqrt{f_{\pi}^2 + 6f_{\pi}^2} = 2f_{\pi} = 2 \times 130 \text{ MeV} = 260 \text{ MeV}$$

إلى طريقة عمل أو تنفيذ البرنامج يتم إدخال عدد صحيح موجب ويتك (n) وينطج قيمة (FB) له حيث إذا كانت: (n=3) فإن (FB=3) أو (n=4) فإن (FB=7)

```

void Fibonacci (int n)
{ // Compute the nth Fibonacci number
  If (n<=1)                                +1
    Count++<=endl                          +1
  Else
  { int Fnum1=0, Fnum2=1, Fnum;           ...-2
    For(int i=2; i<=n; i++)                n
      { Fnum=Fnum1+Fnum2;                  n-1
        Fnum1=Fnum2;                       n-1
        Fnum2=Fnum;                         n-1
      }
    Count++<=Fnum<=endl                    +1
  }
}

```

من حصان<sup>١</sup> وهو ابن القائل للكتاب بالقطر {٥}

معادلات الخوازي . لتطلب الطر-مجموع معادلات حركته بخبر أقدم نوع معادلات (Form1, Form2, Form3) وعنوان للعودة وفي خوارزمية لا يعتمد على خصائص المعادلات  
أي إلى

$$S_{\text{chromatic}}(n) = 0$$

## تعريفات

بعد ان عتدوا حقائق التحليل تعقدت الفرضيات بالبرهان وحالف الوجود شرط مقتضى .  
 التحليل الزماني . عتدوا (1) (2) في تعقدت للوقت (عدد الخصومات ) هي (2) .  
 حالة الثانية عتدوا (1) في عدد الحظوظ هي (4n+1) وكما يرى .

$$T_{\text{Algorithm}}(n) = \begin{cases} 2 & \text{if } n \in (0,1) \\ 4n+1 & \text{if } n > 1 \end{cases}$$

ولنعم بمتية حساب عدد الخطوات في الاجراء البرمجية التالية

```
int i = 1
while (i <= n)
{
    x += x,
    i += 1
}
```

نلاحظ ان الراس (while) تأخذ (n-1) من الخطوات مع مراعاة الانسيب إلى بداية الحداث المستخدم (i) والرعدة المسطرة له.

بما في جزء (Do while) هذا نلاحظ ان (while) والخطوات الحثصة به فعدد (n) من الخطوات

```
int i = 1
do
{
    x += x,
    i += 1,
} while (i <= n),
```

هنا : أوجد متوسط الخزين و، يوافق لمتسلسلة حساب ما يعنى بالبرمططاب المتسلسلة (Prefix Average) لمتتابعة من الأعداد

يفكر برصيح التسلسلة كالآتي: بما كان يجب مصفوفة متية ولكن (X) مخصصة بحري (n) من الأعداد، مصفوفة مثل الخطوات حساب مصفوفة متية هي (A) حيث ان يحصر (A1) نفس البرمط قيم الحاضر من (X[0] ... X[i]) لتيح (i=0 1, ..., n-1) في أنه

$$A[i] = \frac{\sum_{j=0}^i x[j]}{i+1}$$

**Algorithm Prefix Averages(x).**

**Input** An n-element array x of number

**Output** An n-element array A of number

That A[i] is the Average of elements X[0], ..., X[i]

```

For i ← 0 to n - 1 do      n + 1
a ← 0                      n
For j ← 0 to 2 + ... +  $\sum_{i=1}^n (i + 2)$ 
a ← a + x[j]               $\sum_{j=1}^n (i + 1)$ 
and for j
A[i, j] ← a * (j + 1) + x
and for i
return array A ... n

```

تعقيد الخرج: تتطلب المصفوفة ذاتها خزانة مفرقة هي للمصفوفات  $(a[n], n, j)$  و  $(a[n], n, j)$    
 (معرفة) وهو أمر ثابت لا يعتمد على خصائص المثال في  $n$

$$T_{\text{merge sort}}(n) = 0$$

تعقيد الوقت:

$$\begin{aligned}
 \sum_{i=1}^n (i + 2) &= \sum_{i=1}^n i + \sum_{i=1}^n 2 \\
 &= \frac{n(n+1)}{2} + 2 * \sum_{i=1}^n 1 = \\
 &= \frac{n(n+1)}{2} + 2 * n = \frac{n(n+3)}{2}
 \end{aligned}$$

ثم ينعس إلى

$$\sum_{i=1}^n (i+2) = \frac{n(n+3)}{2}$$

ونحفظ المعادلة

$$1 + 2 + 3 + \dots + n - 2 + n - 1 + n$$

في الحقيقة الوقت هو:

$$T_{\text{merge sort}}(n) = n^2 + 7n + 1$$

ملاحظه:  $\leftarrow$  (begin, and, else, etc) تعبر موجبات المخرج ولا تتعد أي حيز أو وقت   
 للمعد (تعارف بوجهه).

نلاحظ في تعقيد الوقت لهذه الخوارزمية أصبحت تربيعية فهل يمكن تحويلها إلى خطية ؟   
 نحاول ذلك كما في التالي:

## 1.1 تحليل أفضل وأقرب وأوسط تنفيذ (Best & Worst & Average Cases Analysis,

يجب علينا ان نلاحظ هذا إذا كانت المسألة تأخذ بكثر من حالة وسوف نقوم بالتركيز على الحالة الأسوأ للعناصر لأنها تحوي تعقيدات كثيرة، كما يمكننا التخلص من الصعوبات في الحالات التي تكون فيها التعقيدات المعنوية (مضامين معنوية) غير مناسبة في كفاءة وحدها التحليل عدد الخطوات وذلك من خلال تحديد تلك أو مع من الخطوات.

أو عدد خطوات الحالة الأصل وهو التي عدد من الخطوات يمكن تنفيذها بعدد عناصر معينة (أو) عدد خطوات الحالة الأسوأ وهو أقصى عدد من الخطوات يمكن تنفيذها بعدد عناصر معينة. تأخذ عدد خطوات الحالة المتوسطة من العدد المتوسط من الخطوات التي يمكن تنفيذها على أنه مثله بعدد عناصر.

مثال: // يوجد العنصر الأكبر في مصفوفة بعينه العدد

Algorithm ArrayMax (A,n)  
Input An array A storing n integer  
Output the maximum element in A  
  
a[0] ← CurrentMax  
1 to n-1 do ← For i  
  If CurrentMax < A[i] then  
    A[i] ← CurrentMax  
  Endif  
Endfor  
Return CurrentMax

- الحالة الأفضل تكون البحث في أقصى حالاته عند تكون أول عنصر في المصفوفة هو الأكبر حيث لا يدخل في تنفيذ أي حل (if)
- الحالة الأسوأ يكون البحث في أسوأ حالاته عندما يكون أخر عنصر في المصفوفة هو الأكبر حيث سيتم تغيير القيمة حتى الوصول إلى النهاية
- الحالة المتوسطة حيث تأخذ بعدد الحالة هي (عدد الخطوات لتح الوصول إلى العنصر (1,2,3, ... n) عدد الخطوات لكنه (n))

كما نلاحظ من مستوى التعقيد يكون حسب الحالة ففي الحالة الأفضل تكون أقل تعقيد وفي الحالة الأسوأ تكون أعلى تعقيد.

ملحوظة: // إلى طريقة حساب عدد الخطوات (Step Count) هي طريقة حساب تقريبية وليس دقيقة وهي صعبة نفس سرفا

$$T_{\text{Arithmetic}}^L(n) = 2n + 1$$

$$T_{\text{Arithmetic}}^P(n) = 3n$$

$$T_{\text{Arithmetic}}^A(n) = \frac{\sum_{i=1}^n (2n + i)}{n} = \frac{2n + \sum_{i=1}^n i}{n} = \frac{2n + \frac{n(n+1)}{2}}{n}$$

مع ملاحظة أنه في حاله المبرمجة فإنه يجري للعداد (i) أن يبدأ من الصفر أو الواحد ، أما بد كلف عملية الحساب يبدأ بالعكس أي من نهاية إلى البداية فتكون

$$T_{\text{Arithmetic}}^A(n) = \frac{\sum_{i=1}^n (2n - i + 1)}{n}$$

### 5.1 أصبع التقريبية (Asymptotic notation)

يوجد ثلاث أصبع تقريبية هي

- 1 صيغة الحد الأعلى (Big-Ob)
- 2 صيغة الحد الأدنى (Omega)
- 3 صيغة الحد الأوسط - حد الثابت (Theta)

برهنت عدله تحدد عد الخطوات (Steps Count) على أنه مجموع غلته في الصغرة وذلك احتجاً إلى أصبع تقريبية لتحديد عد الخطوات

#### 1 صيغة الحد الأعلى (Big-O)

ويقصد به أن تعقيدات الخوارزمية أو الوقت ممكن أن تساوي بعد الأعلى أو تكون أقل منه ولا يمكن أن تكون أعلى منه وعصية أصعب منه قد كالأتي

$$f(n) \leq C \cdot g(n)$$

هذه المعادلة تطبق لنا فقط إذا وجد عدلان موحدان لها (C > 0) بشرط أن (C > 0 & n\_0 = 1) حسب التعميد التالي

$$f(n) \leq Cg \quad n$$

$$n \geq n_0$$

نظرية: لا كلف

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n_1 + a_0$$

فلنعلم حدود درجاتها (m) فلن -

$$f(n) = O(n^m)$$

وهذه بعض الأمثلة لتطبيق النظرية

مثال // ما كانت  $3n + 2$  مثل  $O(n)$  (تتبع خرب بوقت) لأي  $n$  تفزع معين إلى الحل ؟

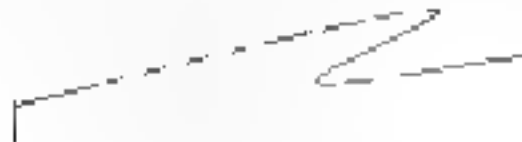
$$3n + 2 = O(n)$$

$$3n + 2 \leq 4n \quad \forall n \geq 2$$

بجميع قيم  $n \geq 2$

حيث إن الكمية  $(4n)$  مثل التواتر أي  $n$  هي ثابت  $g(n)$  بـ  $4$  مثل للوقت  $c$  ، حيث إننا نختار أكبر معامل  $4$  وهو  $(3)$  وأصغره واحد ليصبح  $(4)$  كـ في علقه  $n$   $O(n)$  مثل الحد الأعلى لتتبعات كـ بـ  $4$  .

وهذا يعني بـ  $c$  علقه أفضل التحصيل (النقاط الحقة) في منطقة معينة من  $n$  مع التواتر على شكل الدالة بدون تأثير كما في الشكل رقم (2) التالي



شكل (2) منطقة  $O$  Notation

- تعينه القصة المنظمة
- على الأقل يوجد عملية واحدة

ملاحظة // في الرموز مستخدم الساتر تبدأ من الإحداثيات  $(0,0)$  إلى  $MaxX,MaxY$  لأن سطر الحزب الشرح فقط .

**Example1** Use Big O Notation to analyze the time efficiency of following c++ code of the integer N

```
For(int i=1; i<=N/2; i++)
{
    For(int j=1; j<=N*N; j++)
    {
        // ...
    }
}
```

المعقد بين دوائر For() الخارجية مثلا (N/2) بينما For() الداخلية مثلا (N\*N)  
 $N/2 * N * N$   
 $\frac{1}{2} N * N^2 = \text{big-O} = O(N^3)$

EX N=5;  
 for(int k=1; k<=2; k++)  
 for(int j=1; j<=25; j++)  
 { }

Then  $O(125)$

k	J
1	1
	.
2	25
1	1
	.
2	25

Example2 Use Big-O Notation to analyze the time efficiency of following c++ code of the integer n

```
For(int i=1; i<=n/2; i++)
{ }
For(int j=1; j<= n*n; j++)
{---}
```

$$n^2 + n^2 n = 1/2n + n^2 \quad n + n^1$$

$$n(1+n) \quad \text{Big-O} = O(n^2)$$

ملاحظة: لكسمة (1/2) و (1) مهمالين ، عند هذا كسمة واحدة يتركز لها (C) بوحدة (Big-O)  
 ان ننتج الى فرقت ، فلو فرصد مثلاً:

N=5,  
 For(int i=1; i<=2; i++)  
 For(int j=1; j<= n\*n; j++)

هناك يحتاج (27) تكرار فقط للتكرار وهذا يعني إلى حالة السعيد هذا في أسوأ من الحالة السعيد  
هناك لا تقرب من انه هناك المصعب التالي.

```

k=n
Do
{
  K=k/2
} While (k > 1);

```

هنا عندما تنقص قيمة  $n$  هنا يعني انه الدالة  $n \log n$  أو  $\log n$  يتساوى يمكن  
أن يكون  $O(n^2)$ ,  $O(n)$  هي  $O(n^2)$ ,  $O(n^2)$

When  $n=8$ ,  
Then  $k=8$ ,

k	الحقيقة
8	$8 > 1$
4	$4 > 1$
2	$2 > 1$
1	$1 \leq 1$

إن عدد مرات تكرار هذا المقطع ينقص إلى النصف في كل مرة هذا لأنه ينقسم إلى  
أي إلى (Big-O) له في  $O(\log n)$  وسجده  $O(n^2)$ ,  $O(n)$ ,  $O(n^2)$ ,  $O(n^2)$  رده لها  
تكرر من أقصاه وحدها  $n \log n$  لأننا نصوب في  $n$ .

كما يمكن صياغة المثال بالصورة التالية  
التيك للمعدة التالية

$3n + 2 \leq 4n$   
المطلوب: إذا الصفحة التقريبية أي ثم إذا هذه  $(n)$ .

الحل: أي المعطى يكون شكله كالتالي:

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n_1 + a_0$$

بعد من المعطى فيها أول أو يساوي هذا يعني أن الصفحة في  $O(n)$  (Big-O) المعطى على  
 $n$  هو على أقصاه  $(n)$  هو الواحد وبالتالي يكون الحل الصفحة الأولى.

$$3n + 2 \leq 4n \quad O(n)$$

$$3n + 2 \leq 4n \quad \text{لا}$$

$$n \geq 2 \quad \text{بجميع قيم}$$

n	للطرف الأيسر	للطرف الأيمن	$3n + 2 \leq 4n$	نتحقق
1	5	4	$4 \leq 5$	False
2	8	8	$8 \leq 8$	True
3	11	12	$12 \leq 11$	True
4	14	16	$16 \leq 14$	True



• الصيغة تحقق عندما قيمة الـ  $(n)$  أكبر من أو تساوي (2)

مثال 2: بنا كلف  $10n^2 + 4n + 2$  يعطى  $P(n)$  (التخفيف الحرى ووقت ) لبرنامج معين  
أوجد التعقيد بدلالة الصيغ التقريبية .

$$10n^2 + 4n + 2 = O(n^2)$$

$$10n^2 + 4n + 2 \leq 11n^2 \quad \forall n \geq 6$$

كما يمكن حسابه السؤال بالصورة التالية  
هناك متعددات التالية

$$10n^2 + 4n + 2 \leq 11n^2$$

هذه هي الصيغة المستطرفة رقيقة لـ  $(n)$

الحل// بعد أن استخدمنا أقل أو تساوي فن الصيغة هي (Big-O) و على نفس ذلك  $(n)$  يعطى نفس  
الـ  $(n)$  في الصيغة

$$10n^2 + 4n + 2 = O(n^2)$$

$$10n^2 + 4n + 2 \leq 11n^2$$

جميع قيم  $n \geq 6$

التحقق	$10n^2 + 4n + 2 \leq 11n^2$	الطرف الأيمن	الطرف الأيسر	n
False	$1 \leq 16$	11	16	1
False	$44 \leq 90$	44	90	2
False	$99 \leq 104$	99	104	3
False	$176 \leq 178$	176	178	4
True	$275 \leq 272$	275	272	5
True	$396 \leq 286$	396	286	6
True	$847 \leq 786$	847	786	7

مثال 3: بنا كلف  $(1 - 100)$  أوجد التعقيد بدلالة الصيغ التقريبية ؟

الحل// بن قيمة 100 يعطى  $P(n)$  من الآتي  $P(n)$  يمكن نقوله بالقيمة (1) ، هذا يعنى

$$100 \leq 100 + 1$$

جميع قيم  $n \geq 1$

مثال 4:  $5 + 2^n + n^2 = O(2^n)$  أوجد تعقيد الحرى ووقت بدلالة الصيغ التقريبية ؟

الحل// لاحظ أن هذه البرنامج هناك متعددات أسية وهي على متعددات ذلك إلى .

$$6 * 2^n + n^3 = O(2^n)$$

$$6 * 2^n + n^3 \leq 7 * 2^n \text{ لـ } n$$

تطبيع قيم  $n \geq 4$

ملاحظة: إذا وجد قفص (قفص بحث) يكون على  $n$  موجود في الأمتة السعة فهو لا يؤثر  
ويعتبر في باقي العلاقات صحيحة .

$$\text{مثال 5/6: نحس من صحة المعادلة } 10n^2 + 4n + 2 \neq O(n)$$

نظن:

$$10n^2 + 4n + 2 \neq O(n)$$

$$10n^2 + 4n + 2 \leq 10^4 n$$

تطبيع قيم  $n \geq 10^4$  وهذا غير ممكن

$$\text{مثال 5/6: نحس من صحة المعادلة } 3n + 2 \neq O(1)$$

نظن:

$$3n + 2 \neq O(1)$$

$$3n + 2 \leq 10^4 * 1$$

تطبيع قيم  $n \geq 10^4$  وهذا غير ممكن

لا يمكن أن تكون  $n \leq 10^4$

## 2. صيغة أوميجا الكبرى (Omega $\Omega$ )

ويقصد بها أن عقيدت الحدود أو الوقت ممكن أن تكون كبير أو صغير أو متساوي لأخرى وقد  
ممكن أن تكون أقل منه في نسبة الصعابة يتم كتابتها .

$$F(n) = \Omega(g(n))$$

بذلك وفقد إذا كان لابد فبذلك يوجد لها  $(C, n_0)$  بحيث  $(C > 0)$  و  $(n_0 \neq 1)$  فنـ

$$F(n) \geq Cg(n)$$

تطبيع قيم  $n \geq n_0$

نظرية: إذا كانت

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n_1 + a_0$$

متعددة حدود من حيثها  $(m)$  فإن .

$$F(n) = \Omega(n^m)$$



شكل (3) بمسغه Notation  $\Omega$  (يمكن)

- عملية التصاعد غير منقطعة (مستمرة)
- يوجد زاوية  $(0)$  وب قيمة
- ربما المعنى جيد عند  $0$  عند  $[0, 2]$  وهناك استخدام أي دويقة طسه سيحول أو يغير شكل الدالة تماماً

مثال: ليكن  $3n + 2$  أقل من طيل دالة حزن ووقت البرهان معين أوجد هذه التطبيقات  
بدلالة الصحيح القريبية ؟  
الحل //

$$3n + 2 = \Omega(n)$$

$$3n + 2 \geq 3n \quad \forall$$

جميع قيم  $n \geq 1$

لاحظ هنا مع  $n$  وهو (3) يعني كك هو أي إلى النايب  $c = 3$  ، كك يمكن بعين الصيغة  
(1)  $3n + 2 = \Omega(n)$  فهي لا تؤثر وتبقى الصيغة صحيحة

كك يمكن صناعه سؤال بالصورة التالية  
نذكر المعتمد التالي

$$3n + 2 \geq 3n$$

المطلوب: إيجاد الصيغة القريبية ثم إيجاد قيمة  $n$

$$3n + 2 = \Omega(3n)$$

$$3n + 2 \geq 3n \quad \forall$$

جميع قيم  $n \geq 1$

n	الطرف الأيسر	الطرف الأيمن	$3n + 2 \geq 3n$	نتحقق
1	5	3	$5 \geq 3$	True
2	8	6	$8 \geq 6$	True
3	11	9	$11 \geq 9$	True

مثال 2/ //  $\Omega(n) = 3n + 2$  ما هي المتحددة ؟

نفس تعريف المتحددة هي  $\Omega$  هذا يعني ان العلاقة هي  $\geq$  لتكون المتحددة -

$$3n + 2 \geq 3n$$

مثال 3/ // نثبت لصيغة التقريبية التالية -

$$10n^2 + 4n + 2 = \Omega(n^2)$$

الخطوة 1// نكتب متحددة هذه الصيغة علماً ان  $\{C=1\}$

تجرباً ، يف ان الصيغة هي الحد الأدنى هذا يعني ان الثابت يجب ان يكون أقل او مساوي للحد الأدنى اي  $C=1$

$$10n^2 + 4n + 2 \geq 10n^2$$

بحيث  $n \geq 1$

مثال 4// // نثبت صحة هذه المعادلة  $\Omega(2^n) = 6 + 2^n + n^3$

خطوة 1//

$$6 + 2^n + n^3 = \Omega(2^n)$$

$$6 + 2^n + n^3 \geq 6 + 2^n$$

بحيث  $n \geq 1$

3- صيغة الحد الأعلى  $\Theta$  (Theta)

تستخدم الصيغة لتلخيص صلب المتحددة التي كتبت سابقاً

$$F(n) = \Theta(g(n))$$

إذا ربطنا إذا وحدث تتوافق العوحد التالية  $(C_1, C_2, C_3)$  نكتب يكون -

$$C_1 g(n) \leq F(n) \leq C_2 g(n)$$

$$n \geq n_0$$

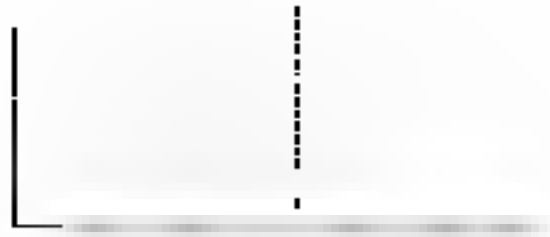
بحيث  $n_0$  نقيم

نظريه 1// إذا كانت

$$f(n) = a_n n^n + a_{n-1} n^{n-1} + \dots + a_1 n_1 + a_0$$

متحددة حدود مرتبتها  $(n)$  فإن -

$$F(n) = \Theta(n^n)$$



#### شكل (4): Notation: $\Theta$ (ثـ)

- كل زيادة تغير الشكل تماماً
- نفس الطول وكل مرة تحذف أو تضيف

مثال: أجب صحة العبارة  $3n + 2 = \Theta(n^2)$  بالعبارة بدلالة صحة مقاربه:  
الحل: العبارة صحيحة

$$3n + 2 = \Theta(n^2)$$

$$\text{لأن } 3n \leq 3n + 2 \leq 4n$$

$$\text{بجميع قيم } n \geq 2$$

لاحظ هذا من تحديد صحة العبارة (2) يكون تصلي وليس عشوائي أي أنه (2) هو فرق تحقق  
الصيغة ليبدأ القيمة (1) لا تحقق بالصيغة

مثال: أجب العبارة التالية  $3n \leq 3n + 2 \leq 4n$

الحل: لا فإنه يوجد غير على وهم مطبق وهذا يعني أنه يجب استخدام صيغة  $\Theta$

$$n \log n = \Theta(n^2)$$

وبه في التواب (C) (C) أكبر من الصغر هذا يعني تحقق الشرط الأول سلك نقوم بتطبيق  
الطريقة

التحقق	الطرف الأيمن	الطرف الأيسر	الرمز	n
False	3	5	4	1
True	6	8	8	2
True	9	11	12	3
True	21	14	16	4

سنتج في الحل الصحيح أنه من  $n \geq 2$

مثال 3// بين ان الصيغة التالية صحيحة

$$10n^2 + 4n + 2 = \Theta(n^2)$$

الحل // بين اني قيمة  $\Omega$   $n$  بصريا بالتوازي

$$10n^2 \leq 10n^2 + 4n + 2 \leq 11n^2$$

لجميع قيم  $n \geq 5$

لاحظ انه يجب ان نضع اقل قيمة لـ  $(n)$  في الحدتين ونقفه الاكل لـ  $(C)$  نضع في النهاية اليسرى والاكبر في الجهة اليمنى

n	تصرف الايسر	الارسط	تصرف الايمن	التحقق
1	10	16	11	False
2	40	50	44	False
3	90	104	99	False
4	160	178	176	False
5	250	272	275	True

ملاحظة// ان صيغة الحد  $\Omega$  على الاقل هي للصيغة الاكثر دقة والتحقق عندما يكون  $\Theta(n)$  هو الحد الاعلى والاقل للدالة  $R(n)$ .

تمرين 1// برهن ان العلاقات التالية صحيحة

$$5n^2 - 6n = \Theta(n^2) \quad (C_1=5, C_2=11)$$

$$38n^2 + 4n^2 = \Omega(n^2) \quad (C=7)$$

تمرين 2// برهن ان العلاقات التالية غير صحيحة

$$10n^2 + 9 = \Theta(n)$$

$$n^2 + \log_{10} n = \Theta(n)$$

وهذه بعض حلول الامثلة السابقة بطريقة الصحيح الخوارزمية.

$$S_{avg}(a, b, c) = \Theta(1)$$

$$T_{avg}(a, b, c) = \Theta(1)$$

بدا ان صيغة الحد  $\Omega$  على تساوي صيغة الحد الاقل فيمكن حينئذ استخدام صيغة  $\Omega$  وبالعكس من ذلك فانه عند استخدام صيغة  $\Theta$  فانه يمكن استخدام صيغة  $\Omega$  او  $\Theta$

$$S_{avg}(n) = \Theta(1)$$

$$T_{avg}(n) = \Theta(1)$$

$$\begin{aligned}
S_{\text{avg}}(n, m) &= \Theta(1) \\
T_{\text{avg}}(n, m) &= \Theta(m \cdot n) \\
T_{\text{avg}}(n, m) &= \Theta(n^2) \quad m=n \text{ حالة}
\end{aligned}$$

#### 6-1 أوضاع الصيغ المتكررة لتقريب (The Times Of Recursive Notation)

يمكن توضيح الصيغ التي تستطيع من خلالها تحديد وقت التنفيذ بالمعادلة التالية  
 $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$

- $O(1)$  تعني أفضل من باقي الصيغ لأخرى ولكن بالمعنى البقية للصيغ هي  $O(\log n)$  في أفضل من باقي الصيغ التي بعدها
- الخوارزميات التي لها تعقيدات مرتبة أو وقت أكبر من  $O(n \log n)$  بعد خوارزميات غير فعالة وكذلك فإن الخوارزميات التي لها تعقيدات ضيقة أي  $O(2^n)$  تكون عملية إلا عندما تكون قيمة  $n$  صغيرة جداً أي أقل من 40
- ولتوضيح ذلك نقرر من يوجد حلف بعد  $10^6$  نقطة في وقت الحظي عند تكون  
 $P(n) = O(2^n)$

When  $n=10$  then  $f(n)=1$  ms  
 When  $n=20$  then  $f(n)=1$  ms  
 When  $n=30$  then  $f(n)=1$  sec  
 When  $n=40$  then  $f(n)=18.3$  min  
 When  $n=50$  then  $f(n)=1.3$  day  
 $4 \times 10^{14}$  year When  $n=100$  then  $f(n)=$   
 $32 \times 10^{14}$  year When  $n=1000$  then  $f(n)=$

تعتبر معطياتنا، أوجد حل مسألة في وقتنا باستخدام أسلوب التكرار (الزمن عام تقديري)

$$\text{Fibo}(n) = \begin{cases} n & n < 2 \\ \text{Fibo}(n-1) + \text{Fibo}(n-2) & \text{else} \end{cases}$$

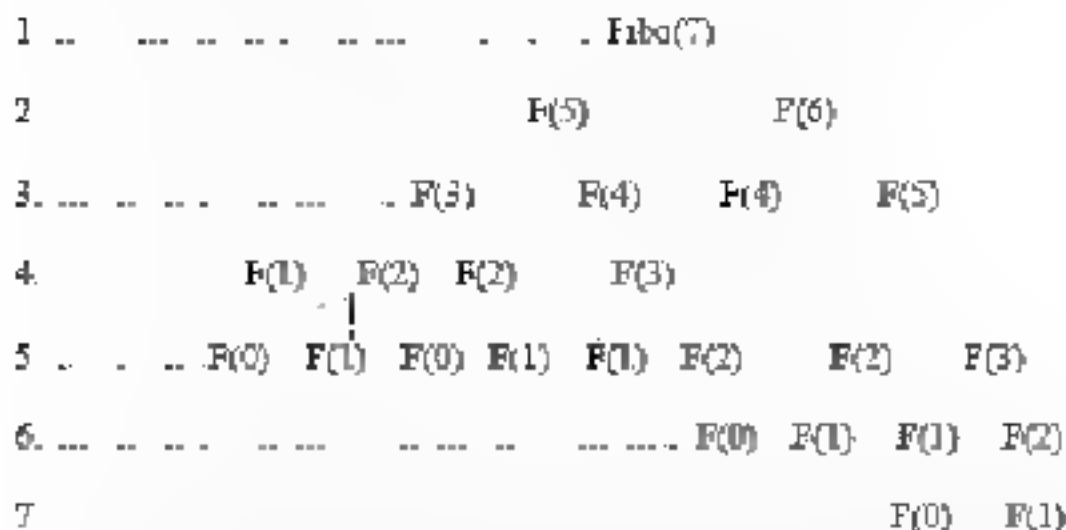
```

int Fibo1 (int n)
{
    if (n < 2) return n;
    Else
        Return (Fibo1(n-1)+Fibo1(n-2));
}

```

بحساب الوقت .

إن عملية حساب التكرارات بسلسلة ليست عام، فكل حالة تحتاج إلى رسم مخطط تدويري  
مكرر، لذلك سوف نرسم شجرة تنشيطات بسلسلة عدداً فيبوناتشي كما في  
التالي:



1 -  $2^n$  يمثل أقصى عدد للعقد في الشجرة، وبما أن كل عقد تمثل استدعاءً من هذه الدالة  
تحت عدد الاستدعاءات يقع بالخطوة له 2 تحتاج إلى حساب التعقيدات الخاصة بكل استدعاء.

### 7.1 الاستدعاءات (Recursion Tree).

ملاحظة: دعنا في الشجرة لتقدير أقصى عدد من العقد هو (  $2^n$  ) حيث إن  $k$  يمثل  
عمق الشجرة ( المستوى الأقصى ) لذلك فإن تعقيدات ديفوناشي تكون:

$$T_{Fib}(n) = O(2^n)$$

وتصبح ذلك:

عدد فيبوناتشي	عدد مرات تنشيط
7	1
6	1
5	2
4	3
3	5
2	8
1	13

نلاحظ أنه في كلتا حالتا  $Fib(30)$  فإن التعقيدات ستكون عدد هائل يقرب 500-000  
تنشيط.





## تعريفات المعية (Practical Complexities)

في تعريف الوقت لبرنامج معين يكون وبصورة عامة في دالة يخصص المثال وهذه الحالة مفيدة جداً في تحديد كيفية تغير متطلبات الوقت بتغير خصائص المثال باستخدام دالة التعقيد الصافي مع زيادة برنامج تقريباً فالحل نفس المعية

ونفترض أنه لدينا البرنامج  $P$  يحوي تعقيداً زائفاً في  $\Theta(n^p)$  وبرنامج  $Q$  يحوي

تعقيداً زائفاً في  $\frac{Q}{\Theta(n^q)}$

السؤال هو أي البرنامج أفضل ؟

ولحل هذه المسألة علينا اتباع التالي

أولاً نكتب برنامجاً جديداً بحيثما يكتب بعد أعلى والأخر يكون التالي هذا يعني أن

من بعد البرنامج  $P$  الذي حجمه  $n$  على هو  $Q(n)$  بعده معية  $Q$  ولجميع قيم

$n_1 \geq n$  حيث  $n$  أقل  $Q(n)$  و  $Q$  أقل الثابت  $C$

من بعد البرنامج  $Q$  الذي حجمه  $n$  على هو  $P(n)$  بعده معية  $P$  وبجميع قيم

$n \geq n_1$

وحيث  $n_1 \leq n$  و  $n \geq \frac{C}{\beta}$  و  $\beta$  و  $\alpha$

في البرنامج  $P$  يكون مربع من البرنامج  $Q$  و  $\beta$

$$n \geq \max \left\{ \frac{\alpha}{\beta}, n_1, n_2 \right\}$$

فإذا فرضنا أن البرنامج  $P$  بعداً في  $n^{10}$  على فاجبة بينما البرنامج  $Q$  بعداً في  $n^2$  و  $\beta$  و  $\alpha$  يكون

$$n \leq 10^4$$



$$M^2 = \frac{t - b}{2t}$$

وفي حالة أنها أصبحت تربيعية فهذا يحصل قطع مكافئ

$$t = a_0 + a_1 n + a_2 n^2$$

أما إذا كانت الخطية فهي  $O(nm)$  لأنه يحصل محطاً ذو صفحة

$$t = a_0 + a_1 n + a_2 n \log n$$

مثال: لقرص قبض، الجارية سوا حقله مخوار رمية يجب التعقيد (Sequential Search)

//منتج

إلا أن هدف من تجريبه هو قياس الحقل الأموي بحيث إن هذه سوار رمية تحتاج وقت قليل جداً راقب في الحاسوب هو أقل من جرد من التلوية بلانك تصحح إن تعدد نواير برمان الوقت كح في جزء البرنامج التالي

```
#include <iostream.h>
#include <iomanip.h>
#include <time.h>
void time Search()
{ // repetition factors
  Long int r[21]= {0,200,000,200,000,1,000,000, . . . 25000};
  Int a[1001],n[21];
  For (int j=1;j<=1000;j++) a[j]=j;
  For (int i=1;i<=10;i++)
  { n[i]=10*(i-1);
    n[i+10]=100*i;
  }
  Cout<<" n n t "<<endl<<endl;
  Cout<<Setprecision(6);
  For (int j=1;j<=20;j++)
  { int h=Gettime();
    For (int i=1;i<=a[j];i++)
    { k=SeqSearch(a,0,n[j]);
      Int h1=Gettime();
      Int t1=h1-h;
      Float t=t1;
      t*=r[j];
      cout<<Setw(5)<<n[j]<<Setw(5)<<t1<<Setw(8)<<endl;
    }
    cout<<" time are in millisecond "<<endl;
  }
}
```



# الفصل الثاني

## الترتيب

### (Sorting)



## 2-2 أنواع الترتيب (Types of Sorting)

- 1 الترتيب الداخلي (Internal Sort)
- 2 الترتيب الخارجي (External Sort)
- \* الترتيب الداخلي يجب ان تكون الذاكرة بحيث يكون حجم البيانات مناسب وليس كبير. ويشمل:
  - 1- ترتيب الاختيار (Selection Sort)
  - 2- ترتيب الفقاعي (Bubble Sort)
  - 3- ترتيب الإدخال (Insertion Sort)
  - 4- ترتيب قنبر (Shell Sort)
  - 5- الترتيب السريع (Quick Sort)
  - 6- ترتيب الأسس (Radix Sort)
  - 7- ترتيب المؤشرات (Pointers Sort)
  - 8- الترتيب الشجري (شجرة البحث الثنائية) (Tree Sort)
  - 9- Topological sorting

\* الترتيب الخارجي هو الترتيب الذي يجب خروج الذاكرة الى أوسط الخزان التقوي عند يكون حجم البيانات كبير بحيث يصعب استيعابها في الذاكرة. هذه عملية الترتيب ويشمل:

- 1 الترتيب بالدمج (Merge Sort)
- 2 الترتيب بالدمج المتوازن أو المتساوي (Balanced Two Way Merge Sort)
- 3 الترتيب بالدمج باستخدام طريقة قسم وضم (Divided & Conquer Merge Sort)

تعتبر الترتيبات المعقدة لا تفضل حواريه الترتيب.

- 1 حجم البيانات المطلوبة: ان كان صغير يكون ترتيب داخلي أما كان كبير يكون الترتيب الخارجي.
- 2- نوع الخزان: ان كان ذاكرة ودمجه يكون الترتيب داخلي أما أنشودة مضطربة يكون الترتيب خارجي.
- 3- درجة ترتيب البيانات: حيث ان ترتيبات الترتيب مرتبة لترتيب بشكل أسرع من الترتيبات غير المتفرقة (مقلدة).

## 2 3 خوارزميات الترتيب الداخلي (Internal Sort):

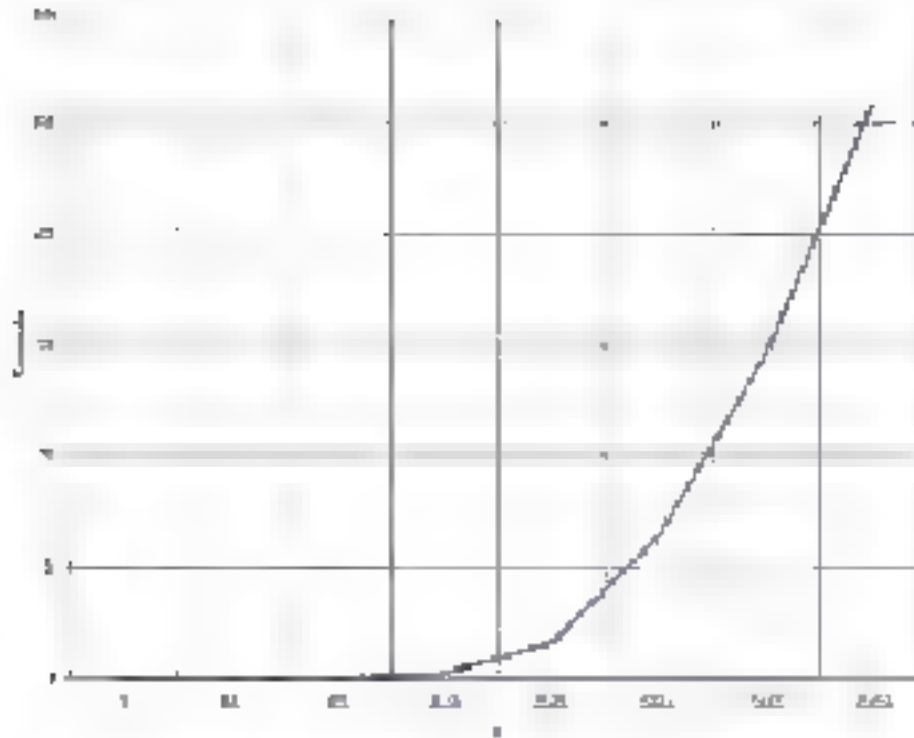
### 1 خوارزمية الاختيار (Selection Algorithm)

ترتيب الاختيار هو خوارزمية لترتيب الأكثر بديهية، و يتم عن طريق البحث في كل عنصر الأكبر من كل العناصر الأصغر و سيوضع في المكان الأخير ثم يبحث عن ثاني أكبر أو أصغر عنصر و الذي يوضع في مكانه في كل المرات الأخيرة و سيخبر حتى يتم ترتيب الجدول بالكامل.





## التحليل التجريبي (Empirical Analysis)



شكل (7) العلاقة بين حجم المدخلات (n) وكفاءة فرز الـ Selection Sort (Selection Sort Efficiency).

والدالة الزمنية التي تقوم بتطبيق هذه الطريقة هي

```
void selectionSort(int numbers[], int array_size)
{
    int i, j;
    int min, temp;
    for (i = 0; i < array_size - 1; i++)
    {
        min = i;
        for (j = i + 1; j < array_size; j++)
        {
            if (numbers[j] < numbers[min])
                min = j;
        }
        temp = numbers[i];
        numbers[i] = numbers[min];
        numbers[min] = temp;
    }
}
```

هذه، دراجع يقوم باستخدام طريقة الترتيب بالاختيار لمجموعة كلمات

```
#include< stdio.h>
main( )
{ Char *z,
  Char *name[] {"ammar", "Fanna", "omar" "shameel", "jamal",
  "saeed", "yousef", "mariam"},
  Int nmax=8;
  Register int i,j,k;
  For i=0; i<=nmax-1 ++i)
  { k=
    z=name[i],
    For(j=i+1, j<=nmax; ++j )
    {if (strcmp(name[j],z)< 0)
      { k=
        z=name[j],
      }
    }
    Name[k]=name[i];
    Name [i]=z,
  }
  For(i=0; i<=nmax; ++i)
    printf("%s\n" name[i])
}
Ahmed
Ammar
Fanna
jamal
Mariam
Omar
Saeed
yousef
```

## 2. خوارزمية الترتيب الفقاعي (Bubble Sort Algorithm)

ترتيب العناصر خوارزمية ترتيب بسيطة لطيفة، وهي تعمل على وضع العنصر الأكبر كفقاعة الهواء التي ترتفع إلى أعلى وذلك بترتيب العناصر بتتابع. أي نقوم بمقارنة العنصرين الأول والثاني، نحفظ العنصر الأكبر، و نبدل الأماكن إذا كانا غير مرتبين. نقوم بهذه العملية إلى آخر عنصر، بعد ذلك نعيد ترتيب باقي العناصر إلى الأخر وهكذا نواصل حتى نصلح عدد وجود عنصر بالحد 1 في صفح لا نقوم بالتبديلات بعد بمر عليه

$$\frac{1}{2} \sqrt{\frac{2}{n}}$$

لترتيب جثث  $A$  بعد  $n$  دفن عدد المقارنات سيكون:

$$\frac{1}{2} \sqrt{\frac{2}{n}} - 1$$

كما عدد التبادلات فهو في المتوسط

تقوم هذه الخوارزمية بترتيب عناصر  $n$  قسماً ثنائياً على عدة مراحل عدد  $n-1$  بحيث يتم وضع عدد واحد على الأقل في رتيبه الصحيح بنهاية كل مرحلة

خطوات تطبيق الخوارزمية

- 1- إدخال الأعداد المراد ترتيبها في مصفوفة  $X$
- 2- استخدام متحول  $switched$  بوضعته على صورت  $(switched=FALSE)$  أو عدم حدوث "تبادل"  $(switched=TRUE)$
- 3- استخدام حلقة خارجية بعدد المراحل  $n-1$  بحيث سرقب الحلقة في حال عدم حدوث تبديل (أو عند هرقبة).
- 4- استخدام حلقة متداخلة تقريفة كل عدد بالعدد الذي يليه حيث يتم تغيير قيمة المتحول  $switched$  إلى  $true$  في حال التبادل.
- 5- استخدام حلقة متداخلة لإظهار ترتيب الأعداد في نهاية كل مرحلة.
- 6- استخدام حلقة لإظهار ترتيب الأعداد النهائي.

وهذا جزء البرنامج بالتطبيق الحصري:

```
#include <iostream.h>
#define MAXNUM 10
enum boolean {FALSE, TRUE},
void main()
{int X[MAXNUM]
  int n, pass, hold;
  int switched=TRUE;
  cout<<"Enter count of numbers:"
  cin>>n;
  for(i=0; i<n; i++)
    cin>>X[i]
  for(pass=0; pass<n-1 && switched==TRUE; pass++)
  { switched=FALSE
    for(i=0; i<n-1; pass, i++)
      if(X[i]>X[i+1])
      { switched=TRUE;
        hold=X[i];
        X[i]=X[i+1];
        X[i+1]=hold;
      }
    }
```

```

for(i=0; i<n; i++)
    cout<<"X["<<i<<"]="<<endl;
    cout<<endl;
}
cout<<"The sort is: "n<<endl;
for(j=0; j<n; j++)
    cout<<"X["<<j<<"]="<<endl;
}

```

نذكره هذه الطريقة لتصنيف أيجاد أصغر العنصر الموجود في قائمة ما، ثم نضعه في مكانه الأول، ونكرر العملية حتى نحصل على القائمة مرتبة.

- 1- First pass
- 2- Second pass

خطى عمل هذه الطريقة هي:

- 1- نأخذ العنصر الأول في القائمة ونضعه في مكانه الأول.
- 2- نأخذ العنصر الثاني في القائمة ونضعه في مكانه الثاني.
- 3- نكرر العملية حتى نحصل على القائمة مرتبة.

مثال: ترتيب العناصر في قائمة ما (Bubble Sort Algorithm).

List	pass(1)	pass (2)	pass (3)	pass (4)
8	8 3 9 7 2	2 2 2	2 2	2
3	3 3 2 8	8 8 3	3 3	3
9	9 2 3 3	3 3 8	8 7	7
7	2 9 9 9	7 7 7	7 8	8
2	7 7 7 7	9 9 9	9 9	9

عدد العناصر  $N = 5$   
 عدد التمريرات  $N - 1 = 4$   
 عدد المقارنات  $N^2/2 = 25/2 = 12.5$   
 عدد التبادلات  $N^2/4 = 25/4 = 6.25$

من هذه الطريقة تكون جيدة إذا كانت العناصر متباعدة عن بعضها وعندئذ ينشأ كثير من المقارنات إلى مساحة تخزين كبيرة، وهذا يعني وقت التنفيذ لهذه الخوارزمية  $O(N^2)$

### 3 هو برزيمه الإدخال (Inserting Sort Algorithm)

نأخذ من هذه الخوارزمية كما يلي

1. نبدأ بالعنصر 2 في القائمة ونقارنه مع العنصر الأول ونضعه حيث الترتيب في مقدمته القائمة وبذلك ترتيباً تصاعدياً.
2. نبدأ بالعنصر 3 ونقارنه مع مقدمته القائمة فنرى بحسب ما على العنصر الأول والثاني ونضعه في الموضع التالي ونسهر بالتسوية حتى الوصول على لقائمة مرتبة

مثال: نأخذ العناصر الآتية بطريقة ترتيب الإدخال (Inserting Sort Algorithm)

8 3 9 7 2 6 4

الحل: يمكن ترتيبها كما في الجدول التالي

List	1	2	3	4	5	6
8	3	3	3	2	7	2
3	8	8	7	3	3	3
9	9	9	8	7	6	4
7	7	7	9	8	7	6
2	2	2	7	9	8	7
6	6	6	6	6	6	9
4	4	4	4	4	4	9

من الملاحظ الإدخال هو عكس برزيم الاختيار لأنه يُنصّب العنصر ويقارنه مع العنصر الذي قبله حيث نأخذ الأول مع التالي والأول مع التالي وهكذا

عدد العناصر  $N=7$

عدد المراحل هو  $N-1=6$

مجموع عدد المقارنات هو  $N^2=49$

مجموع التبادلات هو  $N^2=49$

عقل // البرنامج يقوم بتصنيف حروف هجاء برتتيب الإصافه

```
typedef int tab_entiers[MAX];

void insertion(tab_entiers t) {
    /* Specifications externes */
    int i, p, x;
    for(i = 1 ; i < MAX ; i++) {
        /* position dissertation */
        /* determine p : 0 <= p <= i */
        /* t[p] <= t[i] */
        p = 0;
        while(t[p] > t[i]) p++;
        x = t[i] ; /* t[i] */
        for(j = i-1 ; j >= p ; j--) t[j+1] = t[j] ;
        /* translation t[p+1] vers t[p+1+1] */
        t[p+1] = x ; /* insertion t[i] */
    }
}
```

هذا البرنامج يصنف مجموعة عناصر ، العناصر المتتالية بالرمادي هي العناصر المتضمنة او المتضمنة والتي يراد ترتيبها بين العناصر المكتوبة بالخط العريض bold هي العناصر القريبة في مكانها الصحيح.

تدريج : 29 10 14 37 13

الحل : يمكن توضيحه بالخطوط التالية:

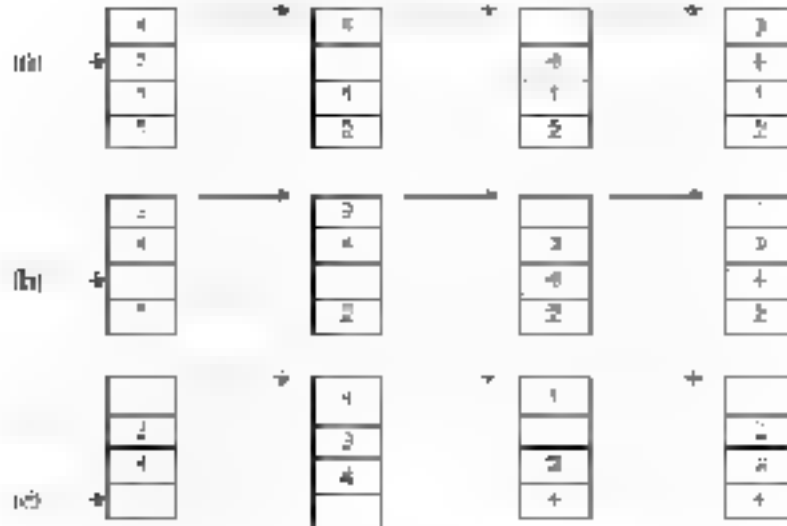
29 10 14 13 37  
 13 10 14 29 37  
 3 10 14 29 37  
 10 13 14 29 37



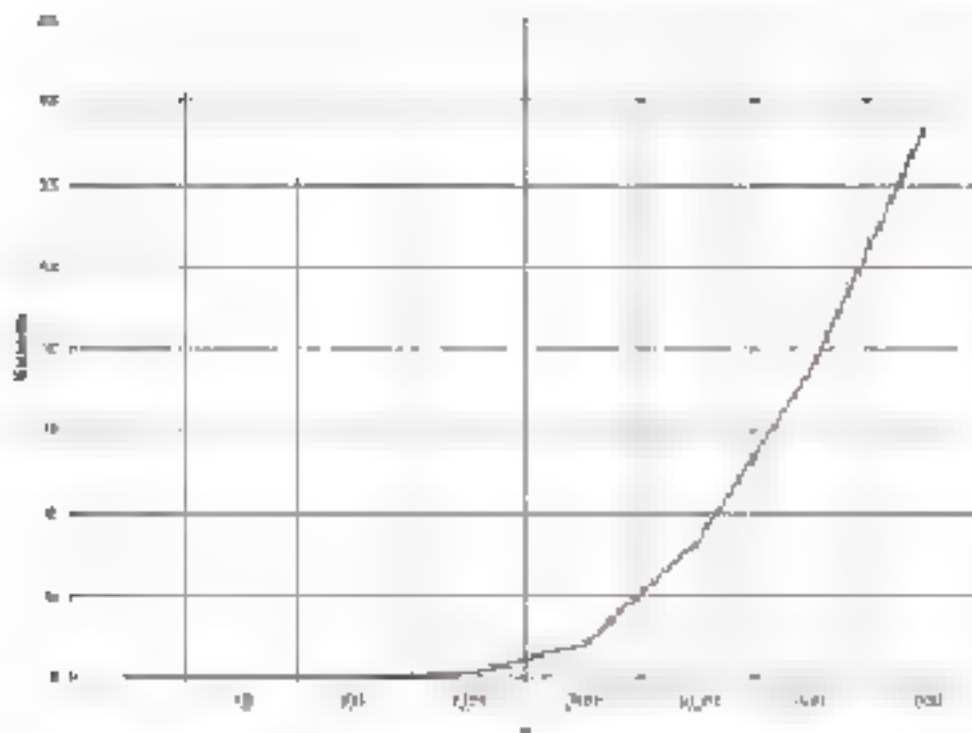


مثال: ديف عنصر الثمانية رتبة (4,3,1,2) باستخدام خوارزمية الإدراج.

الخطوة يمكن ترتيبها في الخطوات (a,b,c) التالية



تحليل تجريبي (Empirical Analysis)



شكل (9) كفاءة خوارزمية الإدراج (Insertion Sort Efficiency)

والحقبة البرمجية الخاصة بتطبيق خوارزمية الإدخال هي :

```
void insertionSort(int numbers[], int array_size)
{
    int i, j, index;

    for (i=1; i < array_size; i++)
    {
        index = numbers[i];
        j=i;
        while ((j > 0) && (numbers[j-1] > index))
        {
            numbers[j] = numbers[j-1]
            j = j-1;
        }
        numbers[j] = index;
    }
}
```

مثال ١٢: برنامج يقوم باستخدام خوارزمية بربيب لإزالة التكرار من مجموعة بيانات

```
#include<iostream.h>
using namespace std;
{
    Char *z;
    Char *name[ ]={'ammar' "Fatima" "omar" "ahmed" "jamil",
                  "saeed" "youcef" "mariam"};

    Int nmax=8;
    Register int i,j;
    For (i=0;i<=nmax-1; ++i){
        z=name[i],
        j= 1;
        While (j>=0 &&(strcmp(z,name[j])<0)){
            Name[j+1]=name[j]
            j
        }
        name[j+1]=z.
    }

    For (i=0, i<=nmax, ++i cout<<" a"<< name[i];
}
```

Ahmed  
 Ammar  
 Fatima  
 jassal  
 Mariam  
 Omar  
 Saeed  
 yousef

### 3 خوارزمية شيل (Shell Sort Algorithm):

يوجد مشكلة في الخوارزمية البعادي هي (في عدد العناصر بزيادة لكن، عنصر في زيادة عدد العناصر في الأعداد في الخوارزمية) هناك أن تكون العناصر في بعد مرتب (في آخر عنصر في الخوارزمية) وأن الموضع الصحيح يجب أن يكون في الموضع الأول، هناك أنه في عدد كبير من المقارنات وهذا يؤدي إلى كثرة الأخطاء.

ولحل لهذه المشكلة من خلال خوارزمتين هي:

1- خوارزمية شيل

2 خوارزمية الترتيب السريع

لكنها تقلص كلاً من. حذف بعض المتقدم الخوارزمية إلى مسافة واحدة، وتحتوي على عنصرين أو أكثر ليس متجاورين وإنما على نفس المسافة لنفسه، ثم ينقسم المسافة الأولية إلى النصف ويحتوي المقارنة أن تستمر ثلثة إلى أن تصبح المسافة مساوية لـ 1، وبذلك يتم ترتيب الخوارزمية

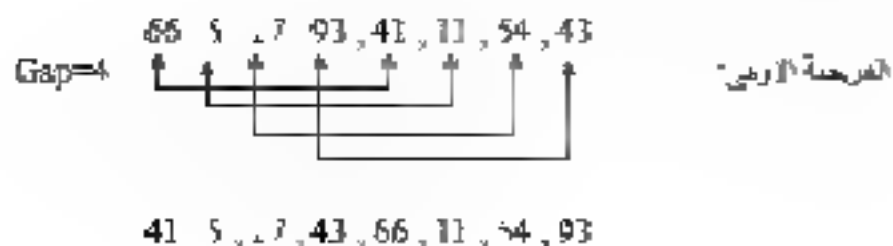
في المسافة الأولية بين عنصرين تدعى فجوة (Gap):

مثلاً: هناك 8 عناصر أولية 66, 5, 17, 93, 41, 11, 54, 43

الحل //

1 بحسب الأعداد المراد ترتيبها  $N=8$

2 نضع المسافة الأولية في النصف 4 Gap



المرحلة الثالثة Gap 4/2 2

41 5, 17, 43, 66, 11 54, 93



17, 5 41, 43, 66, 11, 54, 93



17, 5 41, 11, 66 43, 54 93



17, 5 41 11, 54, 43, 66, 93

المرحلة الثالثة Gap 2/2 1

في هذه المرحلة نستمر بعملية السبيل حتى نحصل على القائمة مرتبة الإرتبة.

5 11 17 41, 43, 54, 66, 93

خصائص تطبيق خوارزمية شل (Shell Sort Algorithm)

- 1 - يزيد كفاءتها كلما زادت عدد العنود
- 2 - لا بد من إجراء عملية الترتيب
- 3 - كفاءة الخوارزمية تعتمد على حجم البيانات المراد ترتيبها
- 4 - لا بد من إجراء عملية الترتيب
- 5 - لا بد من إجراء عملية الترتيب
- 6 - لا بد من إجراء عملية الترتيب
- 7 - لا بد من إجراء عملية الترتيب
- 8 - لا بد من إجراء عملية الترتيب
- 9 - لا بد من إجراء عملية الترتيب
- 10 - لا بد من إجراء عملية الترتيب

توزيع خاصية تطبيق خوارزمية شل (Shell Sort Algorithm)

تستخدم هذه الخوارزمية على الحالة الأكثر تعقيداً في حالة الترتيب

تقدير الأول لتقدير معدل Gap حيث  $Gap = 1.72^{\log(N^{1/3})}$

في المثال السابق استخدم

$N = 8$

المعيار الثاني - تقدير أقصى قيمة لمعدل الوقت (Time average) حيث -

$$T_{avg} = \frac{N^2 \log(N)}{8^{(N/2)}} = \frac{8^2 \log(8)}{8^{(8/2)}} = \frac{64 \log(8)}{8^4} = \frac{64 \times 3}{4096} = \frac{192}{4096} = \frac{3}{64}$$

مثال: ارباب عناصر القائمة (3, 5, 1, 2, 4) بطريقة شبيهة باستخدام فجوة عدد (2) مرة واحدة (1) ؟

الحل: يمكن ترتيبها كما في خطوات الآتي:

(a) Gap=2

3	5	1	2	4
4	3	5	1	2

1, 2, 3, 5, 4

نفس الفكرة فكلنا Gap=1

1	2	3	5	4
4	3	5	1	2

1, 2, 3, 4, 5

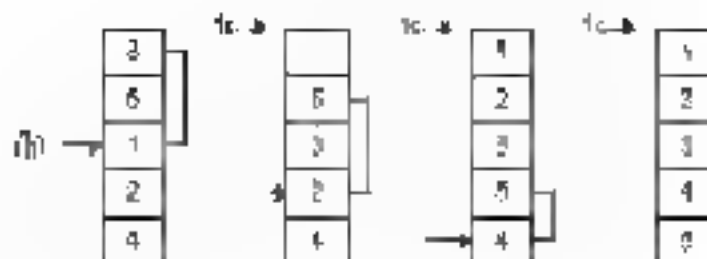
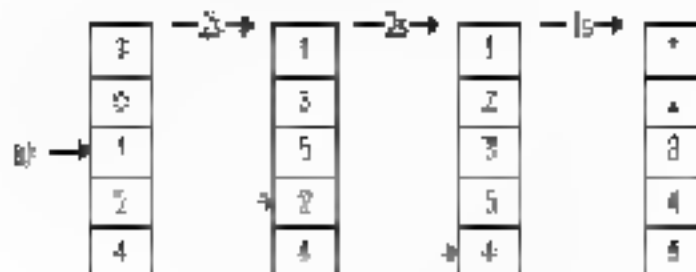
(b) Gap=1

3	5	1	2	4
4	3	5	1	2

نستقر بهذه الترتيب حتى فاصل على قائمة مرتبة

1, 2, 3, 4, 5

ويمكن تمثيل هذه الترتيبات من عناصر القائمة بنفسه بالعصور الصغيرة كما سي .

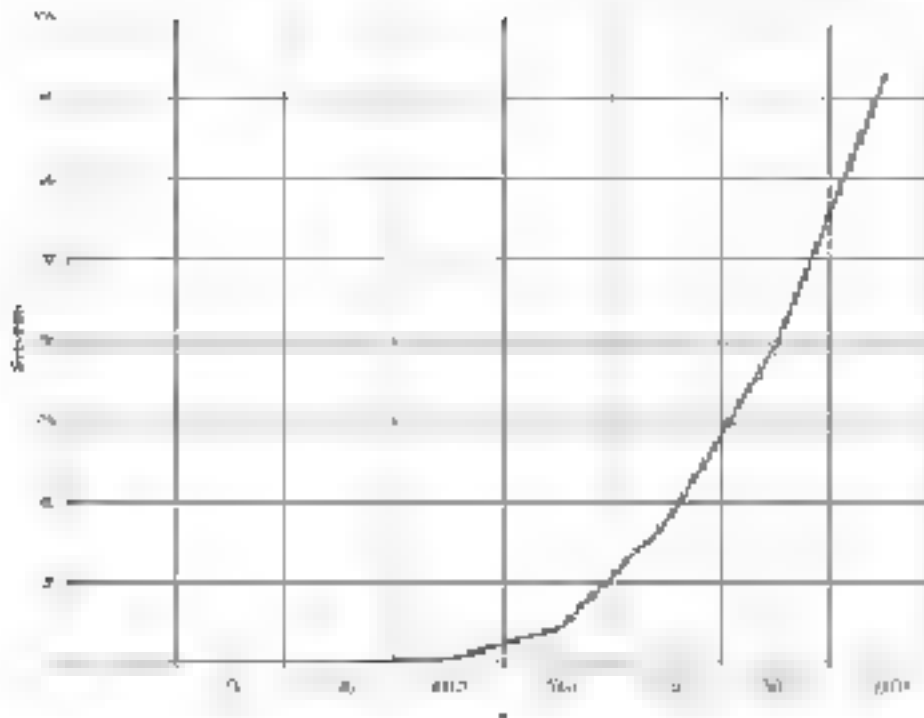


مثلاً: في المثال التالي يستخدم خوارزمية ترتيب شيل (shell) ترتيب مجموعة الأسماء

```
#include< stdio.h>
main()
{
    Char *name[ ]={ 'anwar' "Fatima", "omar" "ahmed", 'jamil'
                    "saeed" "yousef", "mariam"},
    Int m[ ]={9,5,3,2,1},
    Int nmax=8;
    Register int a,b,c,t,v;
    Char *z;
    For (v=0;v<=5; ++v){
        c=m[v];
        t=c;
        for (a=t; a<nmax; ++a){
            z=name[a];
            b=a-c;
        }
        If(b==0){
            t=t;
            t++;
            name[t]=z;
        }
        While((strcmp(z,name[b]))<0)&&(b>=0)&&(b<nmax){
            name[b+c]=name[b];
            b=b-c;
        }
        name[b+c]=z;
    }
    For(a=0; a<nmax; ++a)printf("%s\n", name[a],
}
```

Ahmed  
Anwar  
Fatima  
jamil  
Mariam  
Omar  
Saeed  
yousef

## التحليل التجريبي (Empirical Analysis)



شكل (10): فعالية خوارزمية سبيل (Shell Sort Efficiency)

والجاء البرمجي، فالتحليل التجريبي هو:

```
void ShellSort(int numbers[], int array_size)
{
    int i, j, increment, temp;
    increment = 3; // increment = Gap
    while (increment > 0)
    {
        for (j = 0; j < array_size; j++)
        {
            i = j;
            temp = numbers[i];
            while ((i > increment) && (numbers[i - increment] > temp))
            {
                numbers[i] = numbers[i - increment];
                i = i - increment;
            }
            numbers[i] = temp;
        }
        if (increment > 1)
            increment = increment / 2;
    }
}
```

```

else if (increment == 1)
    increment = 0;
else
    increment = 1;
}
}

```

## ٥. خوارزمية الترتيب السريع (Quick Sort Algorithm)

الترتيب السريع هو طريقة ترتيب من اختراع هور (C A R Hoare) في 1962

### خصائص خوارزمية

تتخذ الخوارزمية في عملها على وضع العنصر الأول (يسمى المؤشر) في مكانه النهائي ثم وضع العناصر الأكبر من المؤشر من جهة اليمين و العناصر الأصغر من جهة اليسار، وتسمى هذه العملية بمرحلة، ثم نقوم بحذف هذه المرحلة بمرحلة مقسمة لكل جهة (اليمين، اليسار) حيث نحدد مؤشر جديدا ونعيد عملية التجربة متكررة هذه العملية إلى أن نحصل على هجوعه مرتبة

إذا لم نحدد المؤشر بطريقة صحيحة، نحصل على الطريقة الأسرع للترتيب في لحظة المعلومات مع تعقيد  $O(n \ln n)$  والتي قد تتحول إلى  $O(n^2)$  في الحالة الاصعب، و في حالة جنوب عناصر مرتبة أصلا، و لكن هذه الحالة بسيطة لأن المجموعة مرتبة أصلا

من الناحية العملية، بالنسبة للتجربة مع عدد قليل لا يجاوز بضع عشرات من العناصر، يتم التجزء عادة إلى الترتيب المباشر الذي يكون أفضل من الترتيب السريع

و بصوره عامة يعتبر الترتيب السريع الأكثر شيوعا (مبسطة) من بين جميع خوارزميات الترتيب حيث أنه مشكلة واحدة يمكن في كافة الحواسيب المؤشر.

### اختيار أفضل مؤشر

عدد متعامل الترتيب السريع مجموعة مرتبة مسبقا، و طريقة اختياره، يستغرق كذا وقت وكثيرا، و ذلك بسبب أن نريد عنصر هو الذي يعتبر هو من الأشياء التي يؤدي إلى عدم عدم مجموعة إلى هجين كذا، و استقر عن المؤشر. لحل المشكلة يتم اختيار عنصر الوسط، كما يمكن اختياره عشوائيا من عنصرين متتاليين حول المركز

تكون لك خوارزمية الترتيب باستخدام مبدأ التجربة حيث نقوم بعد الخطوات التالية.

- 1- تقسم القائمة إلى جزئين حيث نأخذ عنصر للقسمه ولكن في الوسط تقريبا (يسمى  $Q$ )
- 2- نقوم بعملية التفرع بالجدولين بحيث تكون العناصر على جهة اليسار هي الأصغر من  $Q$ ، و إنا نقوم بتقسيم  $Q$  العناصر الموزعة على جهة اليمين فهي الأكبر من قيمة  $Q$



3. نأخذ النصف الأول ونجري عليه عملية مشابهة لذلك سريع مرة أخرى كذلك نأخذ النصف الثاني وهكذا إلى أن تكون جميع العناصر مرتبة

(النصف الثاني (اليمين والأكبر) (X) (النصف الثاني (اليسار و الأصغر )

ملاحظة

\* إذا كانت قيمة (N) هي عدد زوجي فإن قيمة (X) تكون

مثال N=8

$$8 \div 2 = 4$$

نكتبه كالمسار . 1 2 3 4 5 6 7 8

\* إذا كانت قيمة (N) هي عدد فردي فإن قيمة (X) تكون

مثال N=11

$$11 \div 2 = 5.5$$

5 or 6

نكتبه كالمسار . 1 2 3 4 5 6 7 8 9 10 11

مثال 4. رتب العناصر التالية ترتيب تصاعدي باستخدام الترتيب السريع (Quick Sort Algorithm)

20 , 85 , 60 , 75 , 70 , 88 , 50 , 90 , 33 , 95

الحل // نعلم باستخدام الصغير أن التالي

$$X = 10/2 = 5$$

المعرف X : العنصر الموجود في وسط القائمة  
القيمة X = 70

F = Front : متعة القائمة ونكتب بالحد ( )

L = Last : المؤخرة القائمة ونكتب بالحد ( )

المرحلة الأولى X = 5

20 , 85 , 60 , 75 , 70 , 88 , 50 , 90 , 33 , 95

$$I = 1 \quad F = 1$$

$$J = 10 \quad L = 9$$

20 < 95 أي لا يوجد تبديل

20 85 60 75 70 88 50 90 33 95

$$I = 2 \quad F = 2$$

$$J = 9 \quad L = 9$$

85 < 33 أي يوجد تبديل

20 33 60 75 70 88 50 90 85 95

$I=3 \quad F=3$   
 $I=8 \quad L=8$   
 $60 < 90$  أي لا يوجد تبادل  
 $20, 33, 60, 75, 70, 88, 90, 85, 95$

$I=5 \quad F=5$   
 $I=6 \quad L=6$   
 $70 < 88$  أي لا يوجد تبادل

$20, 33, 60, 90, 70, 88, 75, 90, 85, 95$

$I=4 \quad F=4$   
 $I=7 \quad L=7$   
 $50 < 75$  أي يوجد تبادل

$20, 33, 60, 90, 70, 88, 75, 90, 85, 95$   
 $I=5 \quad J=5$   
 $20, 33, 60, 90, 70, 88, 75, 90, 85, 95$

المرحلة الثانية:-

$20, 33, 60, 50, 70, 88, 75, 90, 85, 95$   
 $I=1 \quad J=4 \quad I=6 \quad J=10$   
 $\uparrow \quad \quad \quad \uparrow \quad \quad \quad \uparrow \quad \quad \quad \uparrow$

$X = 33$   
 $70, 33, 60, 50$   
 $\leftarrow X \quad \leftarrow$

$X = 90$   
 $88, 75, 90, 85, 95$   
 $\leftarrow X \quad \leftarrow$

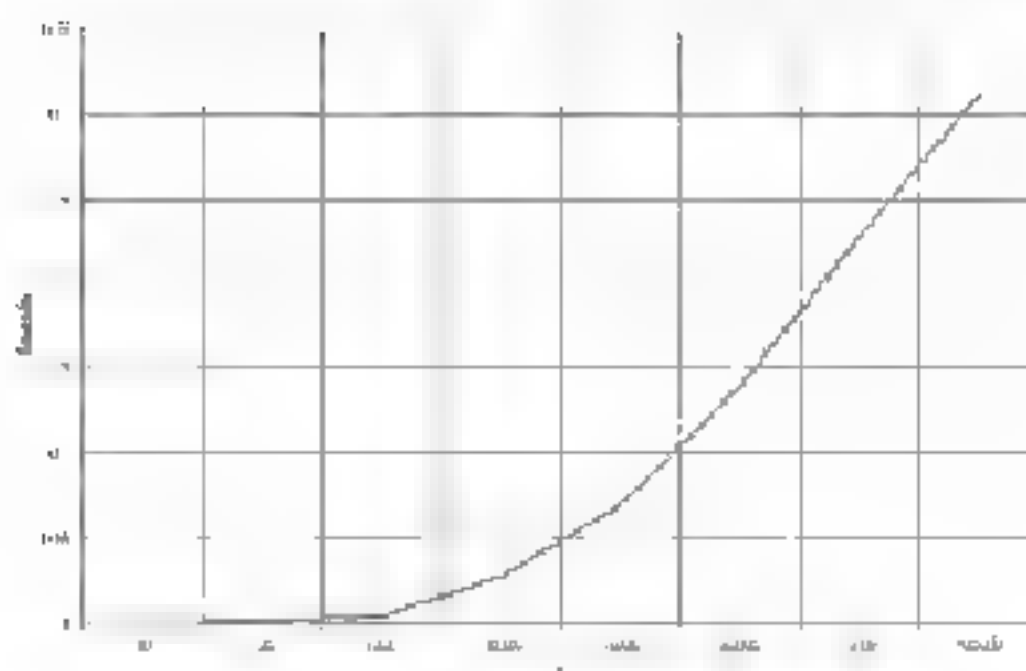
ملاحظة// في خوارزمية الفرز بالقرصعة المتفرقة الكثير من الوقت حاسبه ان نكن عدد  
 العناصر كبير حيث ان  
 $N \log_2 N$  عدد العمل  
 $N \log_2 N^2$  عدد العمل للبيانات

مثال ١٠: رتب عناصر القائمة الآتية (١, ٢, ٣, ٤) باستخدام خوارزمية فرز سريع <sup>\*</sup>

الحل // يمكن ذلك كما في الخطوات الآتية



تحليل تجريبي (Empirical Analysis)



شكل (11) فعالية خوارزمية فرز سريع (Quick Sort Efficiency)

والجزء اليماني الخاص بتطبيق خوارزمية التقسيم السريع هو

```
void quickSort(int numbers[], int array_size)
{
    q_sort(numbers, 0, array_size - 1)
}
void q_sort(int numbers[], int left, int right)
{
    int pivot, l_hold, r_hold;

    l_hold = left;
    r_hold = right;
    pivot = numbers[left];
    while (left < right)
    {
        while ((numbers[right] >= pivot) && (left < right))
            right--;
        if (left != right)
        {
            numbers[left] = numbers[right];
            left++;
        }
        while ((numbers[left] <= pivot) && (left < right))
            left++;
        if (left != right)
        {
            numbers[right] = numbers[left];
            right--;
        }
    }
    numbers[left] = pivot;
    pivot = left;
    left = l_hold;
    right = r_hold;
    if (left < pivot)
        q_sort(numbers, left, pivot - 1);
    if (right > pivot)
        q_sort(numbers, pivot + 1, right)
}
```

## تجربيات أخرى

عند استعمال الترتيب السريع لترتيب هجرو عنه ثابت العناصر كثيره يمكن تغيير تعينه الترتيب عند الوصول الى عطره جريته غير مرتبه عدد العناصر، صغرى 10 عناصر أي أكثر بغير الترتيب بالاختيار مناسب في هذه الحالة

مثال / استخدام برآل تقوم بتقسيم شفهه كثيره بى عجائبه جريته الصغر وشريجه باستخدام حرايهه للترتيب السريع

```
typedef int tab_entiers[MAX];
```

```
int rapideEtape(tab_entiers t, int min, int max) {
    int temp = t[max];
    while (max > min) {
        while (max > min && t[min] <= temp) min++;
        if (max > min) {
            t[max] = t[min];
            max--;
            while (max > min && t[max] >= temp) max--;
            if (max > min) {
                t[min] = t[max];
                min++;
            }
        }
        t[max] = temp;
        return max;
    }
}
```

```
void rapide(tab_entiers t, int deb, int fin) {
    int mil;
    if (deb < fin) {
        mil = rapideEtape(t, deb, fin);
        if (mil < deb > fin - mil) {
            rapide(t, mil + 1, fin);
            rapide(t, deb, mil - 1);
        }
        else {
            rapide(t, deb, mil - 1);
            rapide(t, mil + 1, fin);
        }
    }
}
```

مثلاً برنامج يوضح كيفية امتداد النوار التي تقوم بعملية التقسيم السريع  
(Quick Sort Algorithm)

```
Sort(A)
  Quicksort(A,1,n)

Quicksort(A, low, high)
  if ( low < high)
    pivot location = Partition(A,low,high)
    Quicksort(A,low, pivot location - 1)
    Quicksort(A, pivot location+1, high)

Partition(A, low, high)
  pivot = A[low]
  leftwall = low
  for i = low+1 to high
    if (A[i] < pivot) then
      leftwall = leftwall + 1
      swap A[i] A[leftwall]
  swap(A[low] A[leftwall])
```

الجدول التالي يوضح وقتاً بالثواني لوقت التنفيذ (المتوسط، شئ، والوقت السريع) =

method	statement	average time	worst-case time
insertion sort	9	$O(n^3)$	$O(n^3)$
shell sort	17	$O(n^{2.5})$	$O(n^{2.5})$
quicksort	21	$O(n \log n)$	$O(n^2)$

count	insertion	shell	quicksort
16	39 $\mu$ s	46 $\mu$ s	51 $\mu$ s
256	4,968 $\mu$ s	1,230 $\mu$ s	911 $\mu$ s
4,096	1.315 sec	.033 sec	.000 sec
65,536	436.427 sec	1.254 sec	461 sec

## 6. خوارزمية الترتيب الأسفل (ترقي) (Radix sort Algorithm)

سرع من الترتيب يعتمد على السرعة الموجودة في الارقام وتنقسم إلى حلقاء بحيث ترتب العناصر حسب الارقام (Pockets) على هذه الطريقة يستخدم ما يسمى الخلفاء (Digit) (0-9) بعدد الحلقاء في كل مرحلة بحيث تكون عدد الارقام مساوي في أكبر عدد الارقام في أكبر رقم.

مثال تطبيق // إذا كان لدينا العنصر التاليه (9 7 132)، فإن أكبر رقم يحتوي على 3 مراتب هو (132)

ملاحظة: من هذه الطريقة نعتبر غير عملية وذلك لإعداد متواليات الاختيار في كل مرة بحيث نعتبر (link Queue)، أي أن كل حلقه هي بعيادة طابور عنصر FIFO

مثال: هناك العناصر التالية وبها نستخدم لترتيب بترقي (Radix sort Algorithm).

42	23	74	11	65	58	94	36	99	87
Ordinal									

الحل: يمكن توصيحه بجداول لكل حلقه (مرتبّه) وبك يعني:

0	1	2	3	4	5	6	7	8	9
	11	42	23	74, 94	65	36	87	58	99

1-Pockets1 11,42,23 74,94,65,36,87,58,99

0	1	2	3	4	5	6	7	8	9
	11	23	36	42	58	65	74	87	94 99

2- Pockets2 11,23,36 42,58,65, 74,87 94 99

## 7 ترتيب المؤشرات (Sorting Pointers Algorithm)

يستخدم هذه الطريقة لربط وترتيب العناصر حسب المؤشرات حيث يستخدم فكرة التبادل كما هي الحال التالي .

مثلاً: نأخذ العناصر الآتية بطريقة ترتيب المؤشرات (Sorting Pointers Algorithm)

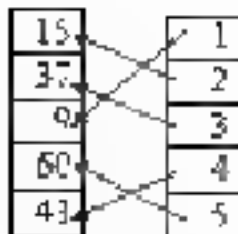
15 , 37 , 9 60 , 43

الخطوة الأولى: يمكن ترتيبها بـخطوات الآتية .

- 1 - صنع العناصر لي خائف
- 2 - صنع المؤشرات



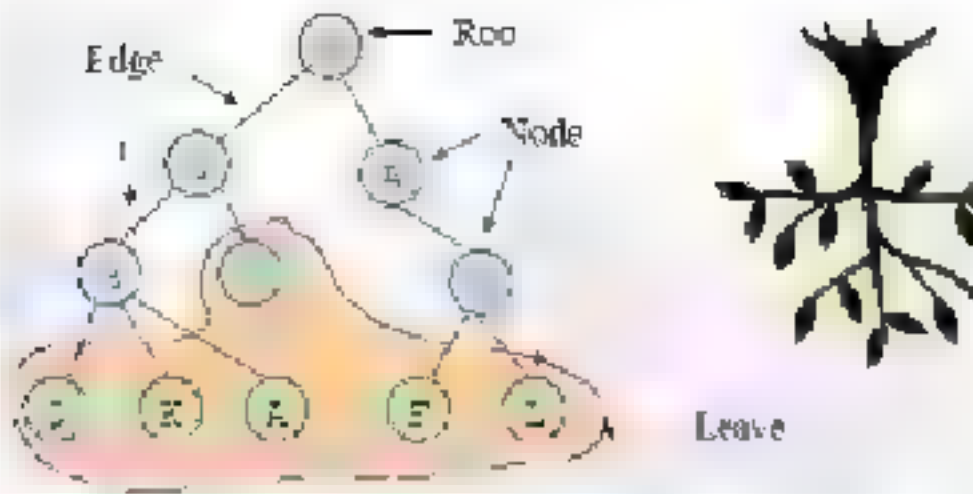
ب - ترتيب المؤشرات



## 8 الترتيب بشجري شجرة بحث الثنائية (Tree for Binary search tree)

الشجرة هي بنية تكون فيها القيمة الأولية لأي عدة أكبر من البنية الأيسر به وبالعكس من القيمة التي هي الأيمن لها. يسمّى من الجذر (Root) والعقد (Nodes) والأوراق (Leaves) وهي خطوات معقدة وكذلك حذف الخطوات معقدة أيضاً ، والشكل رقم (12) الآتي يوضح الهيكل الشجري.





شكل رقم (2) الترتيب الشجري

مثال تطبيق: شجرة بحث ثنائية، يتم بناءها من حساب الأعداد التالية الشجرى



مثال: كود شجرة بحث ثنائية (Tree Binary Search) العناصر التالية بمرأ كل خطوة ؟  
5, 9, 7, 3, 8, 12, 6, 4, 20

الحل: يمكن ذلك من خلال الخطوات الآتية:-  
1- نتخذ العنصر الأول مكون جذر

5

2- نتخذ العنصر الثاني مكون فرع اليمن لأنه أكبر من الجذر

9

(9)

3. نأخذ العنصر الثالث فنكون فرع  $(5)$  العنصر الثاني



4. نأخذ العنصر الرابع فنكون فرع  $(6)$  العنصر



5. نأخذ العنصر الخامس فنكون فرع  $(7)$  العنصر الثالث



6. نأخذ العنصر السادس فنكون فرع  $(8)$  العنصر الثاني



7 - نأخذ العنصر السابع فنكون فرع  $(9)$  العنصر الثالث



8. العنصر الثامن (4) نكرر فرع  $(3)$  العنصر الرابع



9. العنصر 20 يكون فرع يفرع منه (2, 7)



3 4 6 7, 8 9 12 20

بما اننا نستخدم الحذف الخاصة بالاشجار الثنائية فإنه يمكن توضيحها بالطرق الثلاث لاسف:

1- حذف عقدة نهائية (ورقة) - (Leave Delete):

لكي نقوم بحذف عقدة نهائية فائداً يجب ان نغير العقدة وننوي ان نؤشر على عقدة نحتاج



قبل الحذف



بعد الحذف

2. حذف عقدة لها من واحد (One Child Node Delete):

لكي نقوم بحذف الحذف هذه تطبق على شجرة  
1- جعل المؤشر في العقدة يشير إلى العقدة الأولى

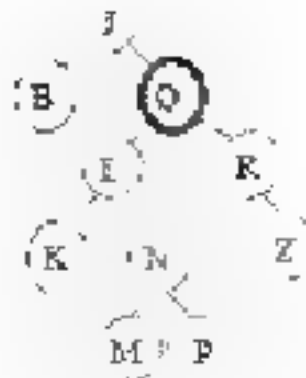
١-ا- نفي العقد المتصوب dispose



١-ب- حذف عقده بعد فرز (Two Childs Node Delete)

- ١- يتم ذلك من خلال الخطوات التالية
- ٢- استفس العقد المطلوب حذفه، بالعثور على العقد الذي له واحد من العقدتين B و C، ويتم استبدال عقدهما من الشجرة
- ٣- الفرعية اليسرى أو الشجرة الفرعية اليسرى للعقد المطلوب حذفه، يتم استبدالها بالعقد المطلوب حذفه
- ٤- نسخة الشجرة الفرعية اليسرى للعقد الذي له العقدتين B و C، يتم استبدالها بالعقد المطلوب حذفه
- ٥- يتم استبدال العقد المطلوب حذفه بالعقد الذي له العقدتين B و C

الشجرة التالية توضح عملية حذف العقد Q واستبدالها بالعقد P







**2** حذف الحقة {ق} وهي حقة بـالقائه (بوالقاء)



3. حذف لفظة (5) وهي عقد الحدي وتمالك مرفق من فقط بنا فلن الاين بيها



#### 4- حساب نقطة (6) : نقطة نهاية



5. حقائق حقة (7) - هذه هي الأمور فقط لنا فهو يريد .



6.  $\left(\frac{1}{2}\right)^n$  is the probability of getting  $n$  heads.



7- حذف عقدة (9) . تلك رقم ليس لها نود يربتها



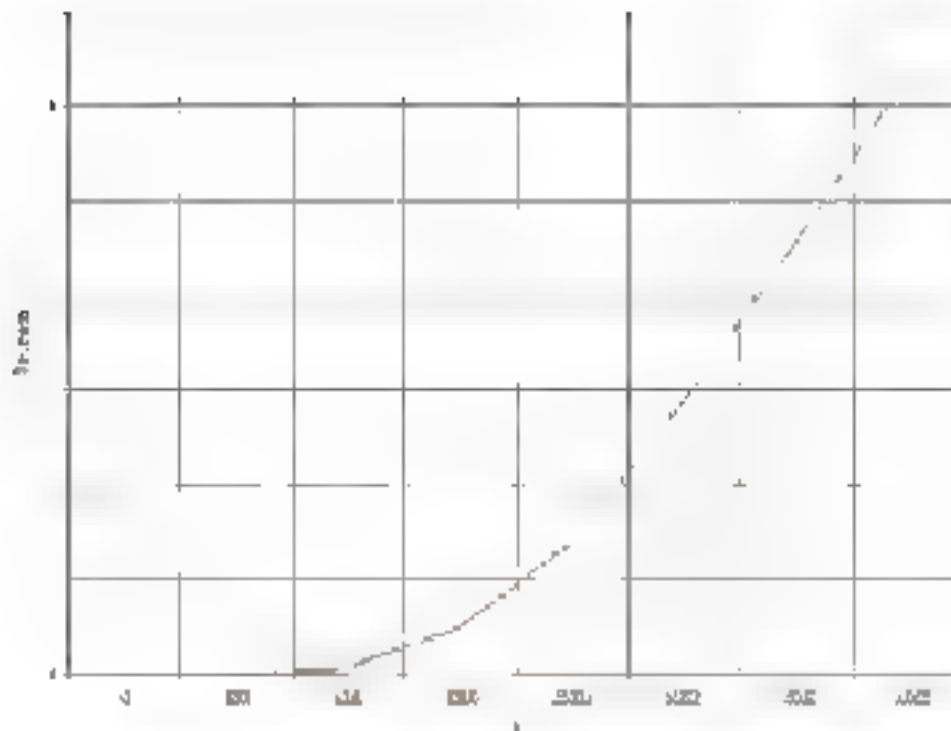
8- حذف عقدة (12) . تلك رقم ليس لها نود



وبوضوح هذه النود يخزنها داخل المكدس كالتالي

3	4	5	6	7	8	9	10	11
---	---	---	---	---	---	---	----	----

نتحصل نتجوبي (Empirical Analysis).



شكل (13): فعالية خوارزمية الترتيب التجريبي (Tree Sort Efficiency)

و جزء البرنامج الذي يقوم بعملية الترتيب للشجرة هو

```
void TreeSort(int numbers[], int array_size)
{
    int i, temp;
    for (i = (array_size / 2) - 1; i >= 0; i--)
        siftDown(numbers, i, array_size);
    for (i = array_size - 1; i >= 1; i--)
    {
        temp = numbers[0];
        numbers[0] = numbers[i];
        numbers[i] = temp;
        siftDown(numbers, 0, i - 1);
    }
}

void siftDown(int numbers[], int root, int bottom)
{
    int done, maxChild, temp;
    done = 0;
    while ((root * 2 <= bottom) && (!done))
    {
        if (root * 2 == bottom)
            maxChild = root * 2;
        else if (numbers[root * 2] > numbers[root * 2 + 1])
            maxChild = root * 2;
        else
            maxChild = root * 2 + 1;

        if (numbers[root] < numbers[maxChild])
        {
            temp = numbers[root];
            numbers[root] = numbers[maxChild];
            numbers[maxChild] = temp;
            root = maxChild;
        }
        else
            done = 1;
    }
}
```



## ٩. خوارزمية الترتيب التوبولوجي (Topological sorting Algorithm)

تستخدم خوارزمية الترتيب هذه فكرة للمصنفات ووضح التيم يانطير، ويمكن توضيحها بالمثل التالي



The graph shown to the left has many valid topological sorts, including

- 1,5,3,11,8,2,10,9
- 7,5,11,2,3,10,8,9
- 3,7,8,5,11,10,9,2
- 2,5,7,11,10,3,8,9

من المصنفات الطوبولوجية هي: وقت التنفيذ خطي، يعتمد بريفيد العقد بين المسارات المختلفة، وقد استخدمت إحدى الخوارزميات كانت من قبل العالم (Kahn 1962)، معتمدة لفكرة الطوبولوجيا

حيث:  $E$

$E$  = الحواف

$Q$  = العنصر

$V$  = الرؤوس

والخوارزمية التي توضح الطريقة هي

```

L ← Empty list where we put the sorted elements
Q ← Set of all nodes with no incoming edges
while Q is non-empty do
  remove a node n from Q
  insert n into L
  for each node m with an edge e from n to m do
    remove edge e from the graph
    if m has no other incoming edges then
      insert m into Q
if graph has edges then
  output error message (graph has a cycle)
  
```

else

output message (proposed topologically sorted order 1)

2-4. في رسم الترتيب الحد هي (External Sorting Algorithms).

1. حوار بعد ترتيب التفرع (Merge sort Algorithm).

هذه، تقوم بترتيب مجموعة عناصر بطريقة التفرع كما في الجدول التالية

مرحلة	T1	T2	T3
A	7 8 8 4	0 9	
B	6		9 8 7 4
C	0 8 7 4	2 6	
D			9 8 7 6 4

وجاء الترتيب الخاص بتطبيق طريقة ترتيب التفرع هو

```
void MergeSort(int numbers[], int temp[], int array_size)
{
    m_sort(numbers, temp, 0, array_size - 1);
}
void m_sort(int numbers[], int temp[], int left, int right)
{
    int mid;
    if (right > left)
    {
        mid = (right + left) / 2;
        m_sort(numbers, temp, left, mid);
        m_sort(numbers, temp, mid+1, right);
        merge(numbers, temp, left, mid+1, right);
    }
}
```

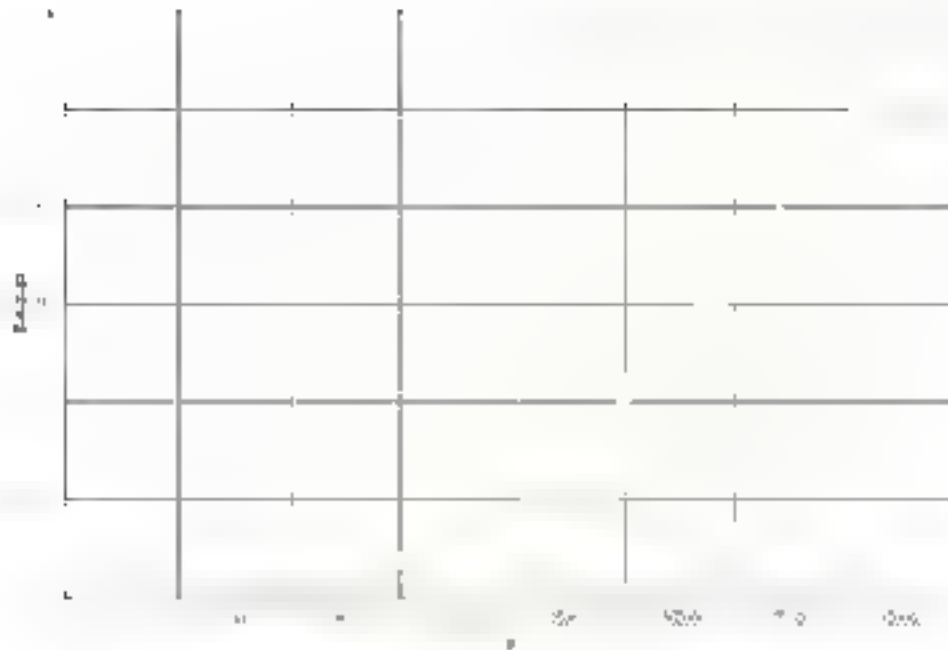
```

}
void merge(int numbers[], int temp[], int left, int mid, int right)
{
    int i, left_end, num_elements, tmp_pos;
    left_end = mid - 1;
    tmp_pos = left;
    num_elements = right - left + 1;

    while ((left <= left_end) && (mid <= right))
    {
        if (numbers[left] <= numbers[mid])
        {
            temp[tmp_pos] = numbers[left];
            tmp_pos = tmp_pos + 1;
            left = left + 1;
        }
        else
        {
            temp[tmp_pos] = numbers[mid];
            tmp_pos = tmp_pos + 1;
            mid = mid + 1;
        }
    }
    while (left <= left_end)
    {
        temp[tmp_pos] = numbers[left];
        left = left + 1;
        tmp_pos = tmp_pos + 1;
    }
    while (mid <= right)
    {
        temp[tmp_pos] = numbers[mid];
        mid = mid + 1;
        tmp_pos = tmp_pos + 1;
    }
    for (i=0; i <= num_elements; i++)
    {
        numbers[right] = temp[right];
        right = right - 1;
    }
}

```

## التحليل التجريبي (Empirical Analysis)



شكل (14) كفاءة خوارزمية الدمج (Merge Sort Efficiency)

## 2- خوارزمية دمج الدمج المتوازنة (Balanced Two-way Merge Sort)

### (Balanced Two-way Merge Sort)

- يمكن توصيف فكر هذه الطريقة بالمثلث الآتي الخاص بملف شجرة كما في الصور التالية:
- 1 تقسم الشجرة إلى نصفين متساويين تقريباً وتسمى A و B ونصع كل عنصر من B مع نظيره الأول في الشجرة A.
  - 2 نقول لنصنع شجرة في الشجرة B مع العنصر نظيره الثاني في الشجرة A ونصنع في الشجرة C بالنسبة.
  - 3 نقول العنصر الثاني في الشجرة B مع العنصر نظيره الثاني في الشجرة A ونصنع في الشجرة D بالنسبة.
  - 4 نكرر الخطوات 2 و 3 لنصنع على عناصر نظيرها 2 في كل من الشجرتين D, C ونصنع العنصر بالنسبة في الشجرتين A, B.
  - 5 نكمل الطريقة نقوم بدمج عناصر الشجرتين A, B حيث عناصرها 4 تكون شجرة ونصنعها في الشجرتين C, D.
  - 6 نعيد الطريقة بدمج عناصر الشجرتين A, B بطول 8.
  - 7 نستكمل بهذا المنهج حتى الوصول إلى شجرة مربعة.

مثال ٨: يقوم بترتيب العناصر  $11$  فيه باستخدام طريقة ترتيب المصنع ذو عنصرين (Balanced Two way Merge Sort).

( 3 , 47 , 33 , 28 , 20 , 63 , 42 , 50 , 2 , 23 , 18 )

الحل: يمكن توضيح ذلك بالخطوات الآتية

1- N = 11      A = 18 , 23 , 2 , 50 , 42 ,  
                   B = 63 , 20 , 28 , 33 , 47 , 3

٢- ترتيب C = 18 , 63      2 18 ,      42 47  
       ترتيب D = 20 , 23      , 33 50      3

تكون فائتين التسميات لتوجيه D والتسميات لفرع C

3-      A = 18 , 20 , 23 , 63      , 3 42 47  
           B = 2 28 , 33 50 ,

4-      C = 2 , 18 , 20 , 23 , 28 , 33 , 50 , 63  
           D = 3 42 47 ,

٥-      A = ٢ 3 18 , 20 23 28 33 , 42 47 50 63 ترتيبه

3- خوارزمية ترتيب المصنع باستخدام طريقة قسم وضم

(Divided and Conquer Merge Sort Algorithm)

يستخدم طريقة قسم وضم ولفصل في فرق قسم ، حيث تقسم المصفوفة المصفوفة إلى مصفوفات من التواليف. كل قائمة تكون من عنصرين بعد ذلك تقوم بدمج التواليف التي تم استخدامها ويمكن توضيحها كما في المثال الآتي -

مثال // مستخدم طريقة فرق قسم في ترتيب المصنع المصفوفة التالية-

( 43 , 54 , 11 , 41 , 93 , 17 , 5 , 66 )

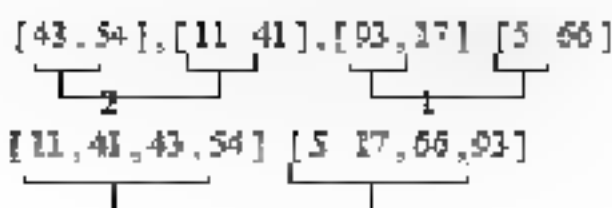
الحل: يمكن توضيح ذلك كما في الخطوات الآتية:

1- نجد عدد عناصر المصفوفة الرئيسية  $N = 8$

2- نصفها إلى طوائف فرعية كل اثنين في قائمة

3- نقوم بدمجها إلى اثنين بترتيب ودمجها

4- نعيد الخطوة السابقة للتوائم المتبقية بترتيبها



43 60 94 43 41 17, 11 5 ختمه من قبله

هذه الطريقة لها مساوئ هي

- 1- إنها تحتاج إلى مصفوفة خزن أمثلية يمكن أن تكون أكبر من المصفوفة الأصلية (  $n^2$  ) على عدد العناصر  $n$  في  $n$  .
- 2- تحتاج إلى مصفوفات فرعية عند كل كبير ، هذه المصفوفات بعض  $n$  في  $n$  مستخرج وهذا يعني أن الوقت المطلوب لإكمال عملية الترتيب كبير نسبة إلى غيرها من الطرق .  
لا تستخرج من ذلك بأن

عدد المراحل  $\log n$  No. Of Pass  
عدد المقارنات  $n \log n$  No. Of Comparison

# الفصل الثالث

## البحث

## Searching

### 3.1 البحث (Search)

البحث هي عملية إيجاد عنصر معين في مجموعة من البيانات فإذا كان العنصر موجود في المجموعة العنصرية تفكير إيجابية وإذا لم يتكون سلبية في حالة عدم وجوده ، ولكن تكون العملية سهلة يحصل أن تكون العناصر مرتبة

### 3.2- البحث التسلسلي (Sequential Search)

في عملية البحث عن عنصر من خلال مسح أو استعراض عناصر القائمة من بدايتها وبالنسبة لحين الوصول لعنصر المطلوب إذا كان موجودا ما في حالة الوصول نهاية القائمة ولم يحصل عملية يعني أن العنصر غير موجود

عدد المقارنات:  $n/2$

وقت التنفيذ:  $O(n)$

مثال: البرنامج التالي يقوم بالبحث عن عنصر من بين مجموعة عناصر باستخدام البحث التسلسلي عفاً عن عدد العناصر (7) والعنصر المراد بالبحث عنه (3) والعنصر هو

data[ ] = 7,4,5,6,3,9,10

الحل://

```
#include<stdio.h>
main( )
{
    int data[ ]={7,4,5,6,3,9,10};
    int nmax=7;
    int key=3;
    printf("%d\n",sqsearch(data,nmax,key));
}
Sqsearch(data,n,k),
int data[ ]
int n,
int k,
{
    Register int i;
    For (i=0;i<=n,++i)
    If (k==data[ i ])return(i+1)
    Return(-1); /* no match exist */
}
```

نتيجة البحث هي أن العنصر (3) موجود في القائمة بالموقع (5)



### 3-3. البحث الثنائي (Binary search).

تقوم فكرة البحث الثنائي على تقسيم المصفوفة إلى نصفين واستبعاد النصف الذي لا يتلقى إليه المصباح key الذي يبحث عنه عن طريق تحديد العنصر الذي يقع في منتصف هذه المصفوفة، ثم يكرر هذا العنصر مع المفتاح الذي يبحث عنه (المصفوفة مرتبة ترتيباً) .  
وال Pseudo code التالي يوضح لك هذه الطريقة.

```
repeat
If ID <= Array[k] then r=k-1
If ID >= Array[k] then l=k+1
until l>=r
If l==j then we found the ID in the array
else the ID is not found
```

مثال : مصفوفة مرتبة ترتيباً أليحيى

```
word[] = {"begin", "const", "do", "end", "if", "odd", "program", "read",
"then", "var", "while", "write"}
```

كيف نكتب code محواري مع البحث الثنائي ؟ كيف يبحث في المصفوفة بترتيب؟  
نقوم بمقارنته بعنصر في منتصف عن طريق نقله لخاصة بمقارنته للمنتصف الحرة في.

```
strcmp (char *str1, char *str2)
```

هذه الدالة تقوم بمقارنته حرفين أو مستطتين حرفيين ونقوم بإرجاع:

التيه (صفر) إذا كانت str1 = str2

قيمة سالبة إذا كانت str1 < str2

قيمة موجبة إذا كانت str1 > str2

و حرم البرنامج الذي نقوم بتطبيق خوارزمية البحث الثنائي ( binary search ) هو .

```
#include "STRING.H"
#include "STDIO.H"
#define max_size 12
//
int binary_search (ID)
char *ID
{
char *word[] = {"begin", "const", "do", "end", "if", "odd", "program",
"read", "then", "var", "while", "write"},
int i=0, j= max_size-1, s, k,
while(s<=j)
{
k=(i+j)/2;
s=strcmp(ID, word[k]);
if (s<=0) j=k-1,
if (s>=0) i=k+1,
}
```

```

if((i-1)>0){
    printf("have found the key (%s) at element %d", word[k] k+1);
    return k
}
return -1,
}
//
void main()
{
    int result;
    char *ID;
    printf("\nFiz. Enter the ID to begin search\nID=");
    scanf("%s",ID);
    result = binary_search(ID);
    if (result==1){
        printf("the key(%s) is not found" ID) }
    getch();
}

```

والتي تطبق خوارزمية البحث الثنائي ( Binary Search ) على مصفوفة م مبع الخسوف  
المتسقة التالية

- 1- الخطوة الأولى: وإذا لم يالسي لا يمكن تطبيق الخوارزمية إلا م كاتب بمصنر  
مرتبة تصاعداً أو تنازلياً أو فمداً على حسب نوع التعلات المخرقة فيها .
- 2- تحديد أول عنصر في المصفوفة المخر 1 ، وآخر عنصر فيها والمعرفات مثلاً 7
- 3- تحديد العنصر الذي يقع في منتصف هذه المصفوفة المخر 4
- 4- م ذلك يمكن تطبيق البحث الثنائي على مصفوفة إلى كمن

أ- إذا كان يساويه مكر قد وجد العنصر الذي يبحث عنه  
ب- إذا كاتب قيمة المفتاح أقل من قيمة العنصر الأوسط في المصفوفة، فإن مخرج البحث  
مط في نصف المصفوفة الأول وننقل البحث إلى نصفه الثاني  
وهذا م ذلك إذا كاتب قيمة المفتاح مكر من قيمة العنصر الأوسط في المصفوفة من مخرج  
أن يبحث مط في نصف المصفوفة الثاني، م مبحث البحث في نصف الأول

حيث تظهر النصف الذي مبحث البحث فيه مصفوفة قلمه يحدد فيها، مخرجها 1, j, & k  
(أي مخرج بتسفيها إلى مخرج) وتطبق نفس الخطوات من 1 إلى 3 فيها، ثم مقرر المفتاح مع  
العنصر الأوسط المخره نفس المخرج مكر في الخطوة 1 إلى 3 السابقة.

- 1- تحديد موقع العنصر الذي يقع في منتصف المصفوفة مخرجاً
- 2- إذا كان العنصر المطلوب (Item) مساوياً لعنصر في الوسط (X) مطي ذلك أنه مسمية البحث  
مع العنصر الذي في الوسط فتمت، إذا كانت أقل من قيمة العنصر في الوسط مسميها  
البحث في النصف اليسرى و أقل من قيمة العنصر (X) مة أنا كان العنصر الذي مبحث عنه  
أكبر من قيمة (X) فمكر البحث في النصف اليمى و لا مكر

3- في الحالتين علاناه تتم المعالجة بنفس الطريقة التي بحث فيها المقارنة للمعادلة بحيث الوصول إلى العنصر المطلوب أي أن  
 معاد العنصر هو  $N$   
 عدد العنصرات هو  $10000$



مثال: نقرر أننا نبحث عن عناصر مختلفة في هذه المصفوفة  
 Array[]: {0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28}  
 و نبحث ترتيب العناصر قبل البدء بالبحث.

الحل: يمكن توضيحه بالبرنامج الآتي

```
#include "STRING.h"
#include "STDIO.h"
#define max_size 15

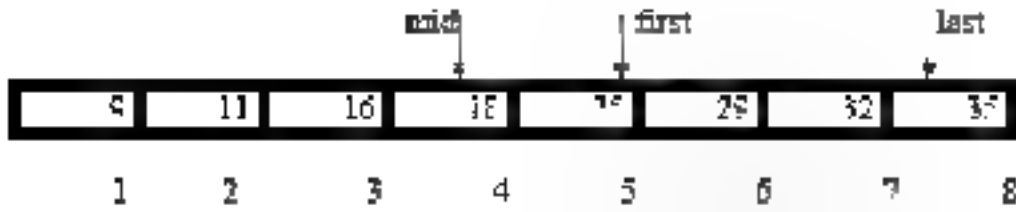
//-----
int binary_search (key)
int key
{
    int NumArray[]={0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28};
    int i=0, j=max_size-1, k=(i+j)/2;
    while(i<=j)
        if (key == NumArray[k])
            printf("we found the key (%d)", key);
            return k;
        }
    else{
        if (key < NumArray[k]){
            j = k;
            k=(i+j)/2;
        }
        if (key > NumArray[k]){
            i = k;
            k=(i+j)/2;
        }
    }
}
return -1
```



2- نجد القيمة الوسطية  $mid = (1+8) \div 2 = 4$

$List[4] = 25$

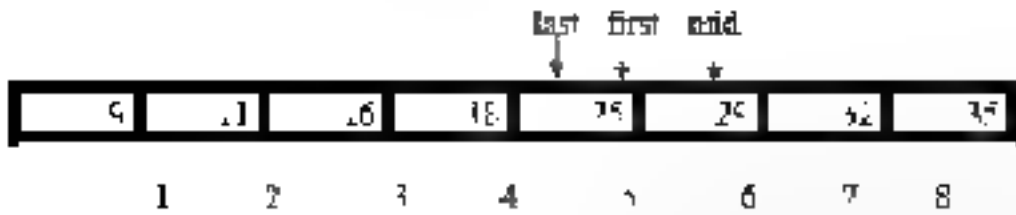
3 من يكرر  $first = mid + 1$



4 نجد  $mid = (5+8) \div 2 = 6$

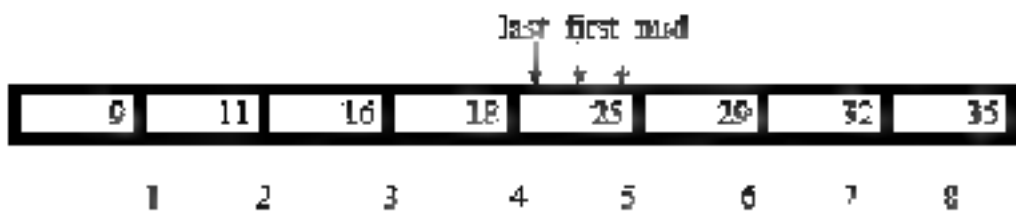
$List[6] = 26$

$Last = mid - 1$



5 نجد  $mid = (5+5) \div 2 = 5$

$List[5] = 26$



النتيجة العنصر موجود بالموقع 5

مثال: استعمل الجدول التالي عن العناصر التالية بطريقة البحث الثنائي

Data[ ]=15,6,0,7 9,23,54 82,101

البحث / للعناصر المراد البحث عنها هي

X=101    x=14    x=82

X=101	Low=i=first	High=j=last	m.d
	1	9	5
	6	9	7
	8	9	8
	9	9	9

$m.d = (low + high) \div 2$

Found=101 في الموضع 9

X=14	Low=i=first	High=j=last	m.d
	1	9	5
	1	4	?
	1	1	1
	2	1	Not found

وسط التوقف هنا  $low > high$

X=82	Low=i=first	High=j=last	m.d
	1	9	5
	6	9	7
	8	9	8

العناصر 82 موجودة في الموضع 8

يفضل إيجاد معدل عدد المقارنات لكلية عناصر باستخدام القانون التالي.

Average of comparison= sum of comparisons/number of elements

	1	2	3	4	5	6	7	8	9
element	15	6	0	7	9	23	54	82	101
comparisons	3	2	3	4	1	3	2	3	4

Average of comparison=25/9=2.77

#### 4- البحث في الشجرة ثنائية (Binary Tree Search)

شجرة البحث الثنائية هي الشجرة التي كل عقدة فيها لكن من عقدها في اليسار عليها واحد من عدد الذي في اليمين منها، الشكل التالي يوضح ذلك.



شكل (S) : شجرة بحث ثنائية

تعد استخدام الشجرة الثنائية للبحث قائمة من الأعداد، يمكن البحث عن عدد ما من خلال بحث العقدة التي بحرفه ونظيره، وفق ما يلي.

إذا كانت العقدة المراد حذفها ورقية، يمكن حذفها دون تغيير بحر في الشجرة.

إذا لم يكن العقدة المراد حذفها ورقية، يجب بعد حذفها وضع مكانها العقدة التي بحرفي العدد التالي وفقاً للترتيب المتصاعدي للأعداد المخرجة في الشجرة.

وجزء البرنامج الخاص بتطبيق البحث في الشجرة السابقة هو

```

#include<iostream.h>
struct nodetype
{
    int k;
    struct nodetype* left;
    struct nodetype* right;
};
typedef struct nodetype* nodeptr;
nodeptr maketree(int x)
{
    nodeptr p;
    p=new nodetype;
    p->k=x;
    p->left=NULL;
    p->right=NULL;
    return (p);
}
  
```

```

void build(nodeptr node, int number)
{
    if(number==node->k)
        if(node->right==NULL)
            node->right=makeTree(number);
        else
            build(node->right,number);
    else
    {
        if(number<node->k)
            if(node->left==NULL)
                node->left=makeTree(number);
            else
                build(node->left,number);
        else
            cout<<"Duplicate number "<<number<<endl;
    }
    return;
}

void search(int key, nodeptr root)
{
    nodeptr p, f, q, rp, s;
    p=root; // p will point to the node
    q=NULL; // and q to its father, if any.
    while(p!=NULL && p->k!=key)
    {
        q=p;
        if(key<p->k)
            p=p->left;
        else
            p=p->right;
    }
    if(p==NULL)
        cout<<"The key does not exist in the tree\n" //leave the tree
        unchanged;
    else // rp will point to the node that will replace node p
        if(p->left==NULL) // node p has right son only
            rp=p->right;
        else
            if(p->right==NULL) // node p has left son only
                rp=p->left;
            else // node p has two sons

```



```

{
    f=p;
    rp=p->right;
    s=rp->left
    while(s!=NULL)
    {
        f=rp
        rp=s;
        s=rp->left; // s is always the left son of rp
    } // now rp is the inorder successor of p
    if(f!=p) // if p is not the father of rp
    {
        f->left=rp->right;
        rp->right=p->right;
    }
    rp->left=p->left
}
if(q==NULL) // if p was the root of the tree
    root=rp;
else
    if(p==q->left)
        q->left=rp;
    else
        q->right=rp;
delete(p);
}

void print(nodeptr p)
{
    if(p!=NULL)
    {
        print(p->left);
        cout<<"p->key:"<<" ";
        print(p->right);
    }
    else
        cout<<" ";
}
}

```

```

void main()
{
    nodeptr tree;
    int number;
    cin >> number;
    tree = make_tree(number);
    while (cin >> number, number != 0)
        balok(tree, number);
    print(tree);
    cout << "Enter the number you want search for and delete" << endl;
    int n;
    cin >> n;
    search(n, tree);
    print(tree);
}

```

اما جزء الخوارزمية الخاص بإصلاحه فقد انشجرة البحث اثنائه فيكون كالآتي:

```

T cc insert(x)
    u ← NIL
    z ← new x;
    while (u ≠ NIL)
        do
            y ← u
            if key[x] < key[y]
                then z ← left[y]
            else z ← right[y]
        if u ← z
    if y ← NIL
    then root[T] ← z
    else if key[z] < key[y]
        then left[y] ← z
    else right[y] ← z

```

y is maintained as the parent of z, since z eventually becomes NIL

The final test establishes whether the NIL was a left or right turn from y.

3. 5- بهید - حواریه لیج - (Complex Of Algorithm Search)

- [illegible]

$$T(n) = O(n \log n)$$

وَلَكِنْ لَقَدْ أَلْهَمَ الْأَعْيُنَ بِحَاسِبِهِ فَقَرَأَ فِيهِ بِحَسْبِ الْإِسْمِ

1- العمل في المؤسسة (Basic Operation)

تعد دراسة عقد حرة دراسة مركز على الأنشطة الأساسية لهذه الدراسة حيث في  
الدراسة التي تلتها الدراسة في عملية التفتيش على التفتيش التي في الدراسة مع رقم  
مخصص للدراسة وكلمة كل عدد تمسك بالدراسة التي كلف الدراسة كلف الدراسة

هذه هي الأخطاء التي يجب تجنبها عند كتابة الرسائل الإلكترونية.

```
int sum = 0;
for ( int i = 1; i <= n; i++ )
    sum += i;
```

نلاحظ أن كل من المتغيرات  $i = 1, \dots, n$   $sum = 0$  في البداية.

تم التعميمات (  $n \rightarrow n+1$  ) مرة بمرات في بعض الحالات مع تحقيق هذه الخصائص.

$$f(\underline{u}) = \underline{z} \quad ?$$

يقولون عن هذه الخوارزمية أنها ذات تعقيد خطي (Linear Complexity) لأننا نحتاج فقط

2. النقيض المتطارب (Asymptotic Contradiction).

بعد إيجادنا في بعض جوانب رعيه وروايتي قتلنا عندنا لافطيات واكثير اهل القصر الصغير ووبعد قتلنا بعد الاثني عشر يوم من القتل في القصر الصغير بعد الاثني عشر يوم من القتل في القصر الصغير

$$f(x) = x^3 + 5x^2 + a - 10$$

يتميز المؤلف 7 كتب هي: 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36. 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53. 54. 55. 56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69. 70. 71. 72. 73. 74. 75. 76. 77. 78. 79. 80. 81. 82. 83. 84. 85. 86. 87. 88. 89. 90. 91. 92. 93. 94. 95. 96. 97. 98. 99. 100. 101. 102. 103. 104. 105. 106. 107. 108. 109. 110. 111. 112. 113. 114. 115. 116. 117. 118. 119. 120. 121. 122. 123. 124. 125. 126. 127. 128. 129. 130. 131. 132. 133. 134. 135. 136. 137. 138. 139. 140. 141. 142. 143. 144. 145. 146. 147. 148. 149. 150. 151. 152. 153. 154. 155. 156. 157. 158. 159. 160. 161. 162. 163. 164. 165. 166. 167. 168. 169. 170. 171. 172. 173. 174. 175. 176. 177. 178. 179. 180. 181. 182. 183. 184. 185. 186. 187. 188. 189. 190. 191. 192. 193. 194. 195. 196. 197. 198. 199. 200. 201. 202. 203. 204. 205. 206. 207. 208. 209. 210. 211. 212. 213. 214. 215. 216. 217. 218. 219. 220. 221. 222. 223. 224. 225. 226. 227. 228. 229. 230. 231. 232. 233. 234. 235. 236. 237. 238. 239. 240. 241. 242. 243. 244. 245. 246. 247. 248. 249. 250. 251. 252. 253. 254. 255. 256. 257. 258. 259. 260. 261. 262. 263. 264. 265. 266. 267. 268. 269. 270. 271. 272. 273. 274. 275. 276. 277. 278. 279. 280. 281. 282. 283. 284. 285. 286. 287. 288. 289. 290. 291. 292. 293. 294. 295. 296. 297. 298. 299. 300. 301. 302. 303. 304. 305. 306. 307. 308. 309. 310. 311. 312. 313. 314. 315. 316. 317. 318. 319. 320. 321. 322. 323. 324. 325. 326. 327. 328. 329. 330. 331. 332. 333. 334. 335. 336. 337. 338. 339. 340. 341. 342. 343. 344. 345. 346. 347. 348. 349. 350. 351. 352. 353. 354. 355. 356. 357. 358. 359. 360. 361. 362. 363. 364. 365. 366. 367. 368. 369. 370. 371. 372. 373. 374. 375. 376. 377. 378. 379. 380. 381. 382. 383. 384. 385. 386. 387. 388. 389. 390. 391. 392. 393. 394. 395. 396. 397. 398. 399. 400. 401. 402. 403. 404. 405. 406. 407. 408. 409. 410. 411. 412. 413. 414. 415. 416. 417. 418. 419. 420. 421. 422. 423. 424. 425. 426. 427. 428. 429. 430. 431. 432. 433. 434. 435. 436. 437. 438. 439. 440. 441. 442. 443. 444. 445. 446. 447. 448. 449. 450. 451. 452. 453. 454. 455. 456. 457. 458. 459. 460. 461. 462. 463. 464. 465. 466. 467. 468. 469. 470. 471. 472. 473. 474. 475. 476. 477. 478. 479. 480. 481. 482. 483. 484. 485. 486. 487. 488. 489. 490. 491. 492. 493. 494. 495. 496. 497. 498. 499. 500. 501. 502. 503. 504. 505. 506. 507. 508. 509. 510. 511. 512. 513. 514. 515. 516. 517. 518. 519. 520. 521. 522. 523. 524. 525. 526. 527. 528. 529. 530. 531. 532. 533. 534. 535. 536. 537. 538. 539. 540. 541. 542. 543. 544. 545. 546. 547. 548. 549. 550. 551. 552. 553. 554. 555. 556. 557. 558. 559. 560. 561. 562. 563. 564. 565. 566. 567. 568. 569. 570. 571. 572. 573. 574. 575. 576. 577. 578. 579. 580. 581. 582. 583. 584. 585. 586. 587. 588. 589. 590. 591. 592. 593. 594. 595. 596. 597. 598. 599. 600. 601. 602. 603. 604. 605. 606. 607. 608. 609. 610. 611. 612. 613. 614. 615. 616. 617. 618. 619. 620. 621. 622. 623. 624. 625. 626. 627. 628. 629. 630. 631. 632. 633. 634. 635. 636. 637. 638. 639. 640. 641. 642. 643. 644. 645. 646. 647. 648. 649. 650. 651. 652. 653. 654. 655. 656. 657. 658. 659. 660. 661. 662. 663. 664. 665. 666. 667. 668. 669. 670. 671. 672. 673. 674. 675. 676. 677. 678. 679. 680. 681. 682. 683. 684. 685. 686. 687. 688. 689. 690. 691. 692. 693. 694. 695. 696. 697. 698. 699. 700. 701. 702. 703. 704. 705. 706. 707. 708. 709. 710. 711. 712. 713. 714. 715. 716. 717. 718. 719. 720. 721. 722. 723. 724. 725. 726. 727. 728. 729. 730. 731. 732. 733. 734. 735. 736. 737. 738. 739. 740. 741. 742. 743. 744. 745. 746. 747. 748. 749. 750. 751. 752. 753. 754. 755. 756. 757. 758. 759. 760. 761. 762. 763. 764. 765. 766. 767. 768. 769. 770. 771. 772. 773. 774. 775. 776. 777. 778. 779. 780. 781. 782. 783. 784. 785. 786. 787. 788. 789. 790. 791. 792. 793. 794. 795. 796. 797. 798. 799. 800. 801. 802. 803. 804. 805. 806. 807. 808. 809. 810. 811. 812. 813. 814. 815. 816. 817. 818. 819. 820. 821. 822. 823. 824. 825. 826. 827. 828. 829. 830. 831. 832. 833. 834. 835. 836. 837. 838. 83

$$f(x) = x^3$$

### 3- التحقق الرسمي في الحالات الأفضل والأسوأ والمتوسطة لخوارزمية (Beast & Worst & Average Complexity)

في المثال السابق دمج يتحقق في نفسه تلقائياً ولكن في معظم المصفوف عند الانتهاء / مخرج  
فقط بقيمة الحجم (n) ليس يتعلق أيضاً بالمتوسط المعالجة لذلك مستخدم حذف البحث  
الحسابي عن العنصر في نفسه من العنصر حيث أنه في هذه الحالة أربعة مقترحة العنصر x الذي  
يبحث عنه مستخدم في array المخزنة في مصفوفة d وطول المصفوفة d = n وإخراج دالة  
العنصر في حال العثور عليه أو إرجاع القيمة -1 في حال عدم العثور عليه

مثال/ جزء من الجي ليصبح خوارزمية بحث عن العنصر x في قائمة مصفوفة d[]

```
int j = 0;
while (j < n && d[j] != x)
    j++;
if (j < n)
    return j;
return -1;
```

مثال / برنامج يحدد مستوى تعقيد Complexity على أن أفضل الأسلوب على الترتيب في البرنامج  
(قد لا تكون جميع هي x دالة n)

```
{
    int mid;
    int first = 0;
    int last = N - 1;
    while (first <= last)
    {
        mid = (first + last) / 2;
        if (list[mid] == target)
            return mid;
        if (list[mid] > target)
            last = mid - 1;
        else first = mid + 1;
    }
}
```

في تعقيد الحالة الأفضل مصفوفة على عدد كبير للعنصر المطلوب إيجاده هو نفسه  
العنصر المتصل به وبالتالي لنحتاج لمقايته وحيداً

أما التعيينات في البحث الأمثل إليها قد يحتاج لتعقيد رياضي لإيجادها فلو كان لدينا مجموعة من الأسماء الموجودة ضمن معملته من كلف في الشكل، والى البحث عن اسم "جورج" فنتأمله في تلك خطوة إلى استقطاب حوالر يمين البحث الثاني، نصلح ستكون بحاجة إلى 7 خطوات لإيجاده باستخدام البحث الخطي.

begin			mid				end
0	1	2	3	4	5	6	7
[	علاء	مكي	موسى	أحمد	فوزي	عزیز	حسن
			begin		mid		end
					begin	mid	end
						begin	end

ويمكن بعد العرف التي تخص هذه الخطوة التالية مع عدد العرف التي يجب علينا فيها تقدير المسألة في الحجم  $\log_2$  وبذلك حتى الوصول لمسألة مرتبة في عنصر واحد. وقد ينتهي بهذا العنصر، وهو العنصر المطلوب.

في المثال المذكور مسألتنا  $\log_2$  (نوعين التاليين) طالما أننا نعلم المسألة الناتجة على 2 عنصر على الأقل (مستوى) جديد. وبالتالي في التوقف في هو التحول هو في المستقر في التقسيم على 2 حتى نصلح العنصر المطلوب في آخر خطوة التقسيم هو 1. نصلي بدلاً من التالي مستطاع بين  $\log_2$  خطوة كسوة بعدد وكلي متطلب سيكون 3 خطوات مسح وجور 8 عناصر.

أما خطوة التقيد الوسطى فانه لا يمكن الوصول لمصلحة نهائية فمثلاً وجود العنصر في مكانه في المسألة هو احتمال وجود العنصر في المسألة الأصلية ضرورياً بحدوث وجود في المسألة الفرعية التالية ضرورياً بحدوث وجود في المسألة الفرعية التي بعدها وهكذا. بالتالي نجد في البحث الثاني عدد يكثير من البحث الخطي وذلك لاقتصره عدد كبير جداً من العناصر في حال كانت  $n$  كبيرة بشكل كبير. مثلاً عندما  $n=100000$  فإن سوا بعد الحصول عليه في حالة البحث الخطي هو  $n=100000$ .

أما في حال البحث الثاني فإن السوا بعدد سيكون  $\log_2(100000)=16$  أي من البحث الثاني وفر لنا 999,984 عملية مقارنة نسبة للبحث الخطي.

الفصل الرابع  
الامتثالية في مسائل تصميم  
الخوارزميات

**Optimization in )  
Algorithms Design  
( Equations**

#### 1.4 المخطط (Graph)

المخطط عبارة عن مجموعة من العناصر التي تمثل نقاط وروابط ( vertices )  
و حيث العناصر التي تربط بعلاقات تسمى حواف (Edges) وهذه العلاقات تمثل بحدود كما في شكل رقم (1.6) التالي



شكل رقم (1.6) يوضح مخطط غير متجه بسيط

$$G = (V, E)$$

$$V(G) = \{1, 2, 3, 4, 5, 6\}$$

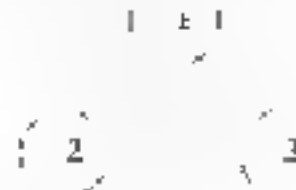
$$E(G) = \{(1, 2), (2, 3), (3, 4), (3, 5), (3, 6), (4, 5), (5, 6), (6, 3)\}$$

حيث  $V$  تمثل مجموعة من النقاط و  $E$  تمثل مجموعة من الحواف

#### 1.5 أنواع المخططات (Type Of Graphs)

يوجد نوعين من المخططات هي

1 المخطط غير المتجه (Undirected Graph) هو المخطط الذي تكون العلاقة بين عناصره غير موجهة أي في الاتجاه تكون غير مهمة كما في الشكل رقم (1.7)



$$V(G) = \{1, 2, 3\}$$

$$E(G) = \{(1, 2), (2, 3), (3, 1)\}$$

شكل رقم (1.7) مخطط غير متجه

## 2- المخطط المتجه (Directed Graph).

هو المخطط الذي تكون الحافة بين عنصرين مرتبة، بمعنى أي من الإحداثيات تكون هوية، ومسبوقة كما في الشكل رقم (18).



$$V(G) = \{1, 2, 3, 4\}$$

$$E(G) = \{(1, 2), (1, 3), (3, 4), (4, 3)\}$$

شكل رقم (18) مخطط متجه

## 2- المخطط المشترك (Undirected Graph & directed Graph)

هو المخطط الذي يحتوي على النوعين كما في الشكل رقم (19) التالي -



$$V(G) = \{1, 2, 3\}$$

$$E(G) = \{(2, 1), (1, 3), (2, 3)\}$$

شكل رقم (19) مخطط مشترك

العنصر هو مجموعة من المتغيرات التي تربط بين عطين في المخطط

مثلاً: في الشكل رقم (20) التالي يوجد عنصر بين نقطتين لهاتين (1, 2) و (3, 2).





شكل رقم (20) بوصف مخطط يجرى بهجوم عد مسارات

الحل :

1. نعتبر الإنعاش للنقطة (1,5) هي (1,3), (3,5) وليس (3,5), (2,3), (1,2)
2. نعتبر الإنعاش للنقطة (1,6) هي (1,3), (3,6) وليس (3,6), (2,3), (1,2)

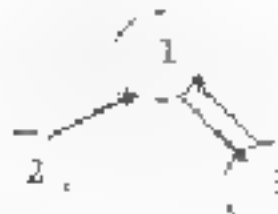
مع ملاحظة انه لا يمكن كتابة المسار باستخدام نفس المسار من المسار { }

#### 3.4 طول المسار (Path Length) :

هي عدد المستقيمت (الخطوط التي تربط نقطتين في المخطط كما نلاحظ في مخطط الشكل رقم

- أ. (3) طول المسار من (2) ويمكن ان يتخذ بطول (3) وهو (3)
- ب. (1,6) طول المسار من (3) ويمكن ان يتخذ بطول (2) وهو (2)

والمعرفة طول المسار وانه عند ان نصل النقاط في نصل عدد (عدد المستقيمت).  
في المخططات المسجة أحيف يوجد أكثر من مسار بين نقطتين وبالتالي فانه نعيد مشكلة تكرر  
في طول المسار (مسار متغير) كما في الشكل رقم (21) التالي.



1. (1,3), (3,4)
2. (1,3), (3,1), (1,3), (3,4)

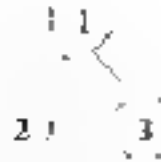
شكل رقم (21) بوصف مسارات مستمرة في اتجاه

### المخطط المتصل (Connected Graph)

هو المخطط الذي يوجد فيه مسار بين أية نقطتين من نقاط المخطط

المخطط غير المتصل (Dis-Connected Graph)

هو المخطط الذي تكون بعض نقاطه غير متصلة ببعضها البعض، بمعنى وجود الشظ (22) التالي يوضح ذلك.



مخطط متصل غير متجه



مخطط متصل متجه



مخطط متصل غير متجه



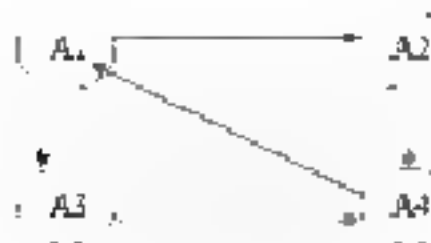
مخطط غير متصل متجه

شكل رقم (23) يوضح أنواع المخططات

يمكننا استخدام المخططات لتحديد أو معرفة أقصر المسارات من خلال إيجاد كل المسارات

وهي ثم بيان الإجمالي فيها

على أن يكون المخطط التالي



المطلوب //

1. توضيح دور  $A$  المخطط
2. نقطي المخطط بمصفوفة
3. نالن قيم المصفوفة
4. نالن نقيد الدخلة والخارجة من كل نقطة
5. نحدد قصور مسلي بين نقطة  $A_1$  ونقطة  $A_4$

الحل //

1. المخطط متجه ومتماثل (Direct Graph & Connected Graph)
2. يمكن تغيير المخطط بالمصفوفة التالية

	1	2	3	4
1	A11	A12	A13	A14
2	A21	A22	A23	A24
3	A31	A32	A33	A34
4	A41	A42	A43	A44

3. يمكن بيان قيم المصفوفة أعلاه كالآتي

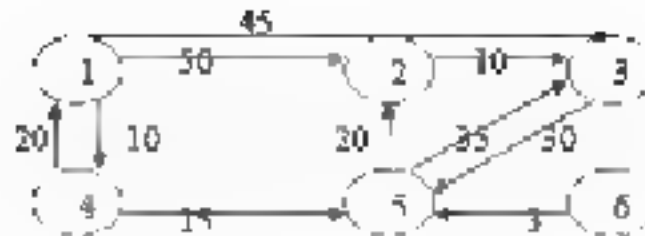
		A1	A2	A3	A4
2	A1	0	1	1	0
1	A2	0	0	0	1
1	A3	0	0	0	1
1	A4	1	0	0	0
		1	1	1	2

4. نوجد مجموع نقيد الدخلة والخارجة من كل نقطة قل  
مجموع النجم لي كل صف يمثل عدد الخطوط الخارجة من كل نقطة  
(Row = Out degree)  
مجموع النجم لي كل عمود يمثل عدد الخطوط الدخلة للنقطة  
(Column = Input degree)

المسارات الداخلية	المسارات الخارجية	اسم النقطة
1	2	A1
1	1	A2
1	1	A3
2	1	A4

5. نوجد قصور مسلي بين نقطة  $A_1$  ونقطة  $A_4$   
a المسار  $(A_1, A_4)$  ،  $(A_2, A_4)$  ،  $(A_1, A_3)$   
b المسار  $(A_3, A_4)$  ،  $(A_1, A_3)$   
وهو قصور المسارات لأن درجاتهما تساوي (2)

مثال 11 شبكة المخطط التالي



المطلوب:

- 1- حدد نوع المخطط
- 2- حدد المصفوفة
- 3- حدد قيم المصفوفة
- 4- حدد العيود الداخلة والخارجة
- 5- حدد السلاسل بين نقطته
- 6- أعط القيم لسلاسل النقطه (1,5) و (1,6) و (2,6)

الحل:

- 1 للمخطط متصل وبعينه
- 2 المصفوفة هي

	1	2	3	4	5	6
1	A11	A12	A13	A14	A15	A16
2	A21	A22	A23	A24	A25	A26
3	A31	A32	A33	A34	A35	A36
4	A41	A42	A43	A44	A45	A46
5	A51	A52	A53	A54	A55	A56
6	A61	A62	A63	A64	A65	A66

- 3- حدد قيم المصفوفة كالآتي

	A1	A2	A3	A4	A5	A6
A1	0	50	45	10	0	0
A2	0	0	10	0	0	0
A3	0	0	0	0	20	0
A4	20	0	0	0	15	0
A5	0	20	30	0	0	0
A6	0	0	0	0	3	0

#### 4. تحديد المسارات الداخلية والخارجية

المسار الداخلي	المسار الخارجي	سم النقطة
20	105	A1
70	10	A2
50	30	A3
10	35	A4
49	55	A5
0	2	A6

#### 5. إيجاد أقصر مسار بين النقطتين

- المسار بين (1,6) هو " لا يوجد مسار بينهما
- المسار بين (1,5) هو (4,5) ، (1,4)
- المسار بين (2,6) هو " لا يمكن التوصل إلى النقطة (6) لعدم وجود مسار داخل إليها

#### 4. طريقة الجشع (Greedy Method)

في هذه الطريقة نستخدم غالباً لحل مسائل الأمثلية (Optimization problems) التي علمه في تكبير (Maximum) شيء معين أو تصغير (Minimum) غير الممكن الشيء ، كما في حلالة الربح أو الخسارة

في هذه المسائل نحاول على العنصر التالية:

- دالة هدف (Objective Function) وهي تمثل دالة يجب أن نحققها - فهدفنا هو إيجاد الحل ممكن ، وسعى أفضل الحلول الممكنة وهو الأفضل .
- في مجموعة القيود (Constraints) ، فإن مجموعة الحلول التي نحققها اختيار ممكن حلول ممكنة (Feasible Solutions) ، وكل الحلول التي يعطي دالة هدف يساوي الحل الأمثل (Optimal Solution)

#### إن المشكلة لهذه الطريقة هي

(يعني الحل الأمثل في طريقة الجشع على مراحل في كل مرحلة يُحدد نفس قرار قبله فليس هناك ملاحقة ، حيث إن التبرير لمبدأ من تغيير الحقائق يجب أن نحقق مثله )

ملاحظة: إن أغلب المسائل التي نطبقها طريقة الجشع تكون من أكثر من واحد إلى الفصل (n input) حيث نهدف إلى إيجاد من التعديل الذي حالياً هو الأفضل ولكن قد يكون هو الآخر قد هذا يعني أنه يوجد مساران أو مشاكل لهذه الطريقة .

#### يوجد نموذجين لطريقة الجشع هما:

#### 1. نموذج المجموعات الجزئية (Subset Paradigm)

في هذا النموذج نبحث عن مجموعة جزئية من المجموعة الأصلية بحيث تكون هيئة ويجب أن نحقق هذه المجموعة قيود المسألة ، وفيه يلي تجريد مبسط أو تحكمي لهذا النموذج:

```

SolType Greedy (type a[],int n)
a[1..n] contains the n inputs.
{solType solution=Empty, initialize the solution
For(int i=1; i<=n; i++)
{ typeX= Select(a);
  If feasible(solution,X)
    Solution= Union(solution,X);
}
return solution;

```

إن المقصود من (Solution) هي مجموعة العناصر التي لا تحتوي أي عنصر في المجموعة (Set S) فهي حالة يحتل بها مجموعة مختارة ونرجع واحد من العناصر لم يتم الاختيار هل إلى الحل هو حل ممكن؟ نعم فإنه يصلح هذا الحل لمجموعة الحلول السابقة وإلى حالة يمكن فإنه يضاف إلى الحل الأمثل

#### 4.5 مسألة حقيبة (Knapsack Problem)

سنأخذ مسألة الحجاب أو حقيبة الظهر كمثال نموذج للمجموعة التجريبية (Knapsack Problem) حيث هناك مجموعة كلف بوصف أي هذه الخفية

مجموعة المسألة (P) من الكيفيات حيث يمكن أن تكون أي شيء واذن حجاب مسالة (C) مقسماً بالكلية غير أن (لا يمكن فتحها بوزن من هذه الكيفيات)

لتكن (I) المجموعة (W) حيث (1 ≤ i ≤ n)

بمسألة الكسر (r) وهو يمثل حل لمسألة حيث (0 ≤ kr ≤ n) يحقق ذاته هي (P(x))

مختار - في الحجاب بحيث نعلم القيمة القصوى أي أنه هناك مسالة (Maximum)

دالة الهدف هي معيار الأمثلة (نفسه الكسر بعد الدالة في معيار الأمثلة أي الحل الأمثل)

$$\text{Maximize } \sum_{i=1}^n P_i X_i$$

ثم طباعة النتيجة (X) حيث بالاعتماد على قيمة النتيجة فإنه يمكن جلاء من الخلفه صنف أو لا صنف

في نموذج المسألة يجب أن نحقق

$$\sum_{i=1}^n W_i X_i \leq C$$

1 أي أي حل يحتوي على المسألة هو حل ممكن

2 أي حل يحقق الكسر بعد زيادة الهدف هو الحل الأمثل

والنتيجة هي

$$0 \leq x_i \leq 1 \quad 1 \leq i \leq n$$

and

$$P_i \neq 0 \quad \forall \quad 0 \leq i \leq n$$

ومتلخص الآن مثالاً لتطبيق هذه الطريقة .

$$P[1, 3] = \{25, 24, 15\}, C = 20, n = 3$$

$$W[1, 3] = \{13, 15, 0\} \quad \begin{matrix} P_1 \\ P_2 \end{matrix} \quad \begin{matrix} 1 & 3 & 1 & 6 & 1 & 5 \end{matrix}$$

في هذه الطريقة تعطى ثلاث جدول عمقهم مختلف باختلاف العمق كما يأتي:  
الجدول الأول : أنه أول حل محتمره يكون هو الحل الأمثل بالقيمة  
الحل الثاني : أنه أول حل محتمره يكون هو الحل الأمثل بالوزن  
الحل الثالث : أنه أول حل محتمره يكون بالاعتماد على سعة تقسم التفاضل على الواحد المقطرة .  
سوف نقوم بعد جدول لتوضيح حلول المعادلة

المعامل	$\sum_{i=1}^n P_i$	$\sum_{i=1}^n W_i$	$x_1 \quad x_2 \quad x_3$
المعادلة الأولى بوزن	20	28 2	$(1, 2/15, 0)$
الوزن الأقل أولاً	20	31	$(0, 2/3, 1)$
المعادلة الأمثل لكل وحدة وزن أي $(\frac{P_i}{W_i})$ بوزن	20	31 5	$(0, 1, 1/2)$

وبالتوضيح هذه النتائج بتطبيق التعبير الأول كالتالي:

$$20 - 18 = 2$$

$$2 - 15 = 2/15$$

$$0 - 10 = 0, 10 = 0$$

هذا التعبير كونه وحدة وأساري (1) بحيث بعد طرح الوزن من التحرك . وهكذا  
سنمر بهذه المعادلة حتى يتم على الحروف  
مع ملاحظة النتائج للخبر : أعلاه بالتحليل المعيار الثالث هو الذي يعطى نتائج الغير قائمة  
ومثالاً لن هي  $(2, 2, 3)$  التبعه له تعال الحل الأمثل.

ولذلك من صحة النتائج يمكننا إجراء عملية ضرب بين قيم المعيار النتيجة والوزن في  
الخاصة بها .

وهذا هو الإجراء المتبع في تنفيذ خوارزمية Knapsack Problem

```

Void Greedy Knapsack (float m, int n)
// p[1..n] and w[1..n] contains profits and weights
// respectively of the n-objects ordered such
that  $p_i/w_i \geq p_{i+1}/w_{i+1}$ .
// m is the Knapsack size and x[1..n] is the solution vector
    n هو قسمة عدد الكتل (m) على سعة الجراب ، ويجمع الكتل في السلة
    منتهية على هذه السلة حيث يصعب الترتيب تنازلي أثناء الحل وبالتالي فإنه من
    الممكن الانتقاء للحل الأول والذي يعال لأنه هو في النهاية هو الحل X يكون فرق
    مهماً أي يعبر كيف صغيره
{
    For(int i=1; i<=n; i++)    x[i]=0.0;
    float U=m;
    For(j=1; j<=n; j++)
    {
        If (W[j] > U) break
        X[j]=1.0;
        U = U - W[j];
    }
    If (j<=n)    x[j]=U/W[j];
}

```

مميزات خوارزمية

تطلب هذه الخوارزمية بعض المساحة من وقت ترتيب الكتل ترتيباً  $O(n)$  من الوقت فقط حيث إننا نحقق ترتيباً بوقت يكون صفياً ووقت الترتيب حيث أنه في حالة استخدام خوارزمية ترتيب على وقت التحقيد هي  $n \log n$  لأنها تكون دائماً وتعطي حلاً أمثل

ملاحظة: يوجد خوارزمية أخرى (Knapsack 0/1) حيث أنها هي نفس النتيجة أو لأفضلها وبالتالي فإنه يجب حذف نظرية (If) الأخيرة من هذه الخوارزمية وفي هذه الحالة لا يكون هناك صغائر للحصول على حل أمثل



## 2. نموذج الترتيب (Ordering paradigm).

في هذا النموذج يتم اتخاذ القرارات باعتبار المسلمات بتوقيت معين بحيث كل قرار يتم اتخاذه باستخدام معيار التفضيل والذي يمكن حسابه من خلال تفرعات المسألة السابقة

ملاحظه ، في مثال الترخيص هناك قبل واحد يحصل بمرور في تغير الحل

لترتيب هذا النموذج سوف نحدد مسألة بمعنى مسألة الحفاظ الترخيص الأمثل **Optimal Merge Pattern** حيث تتلخص هذه المسألة بالمعنى التالي

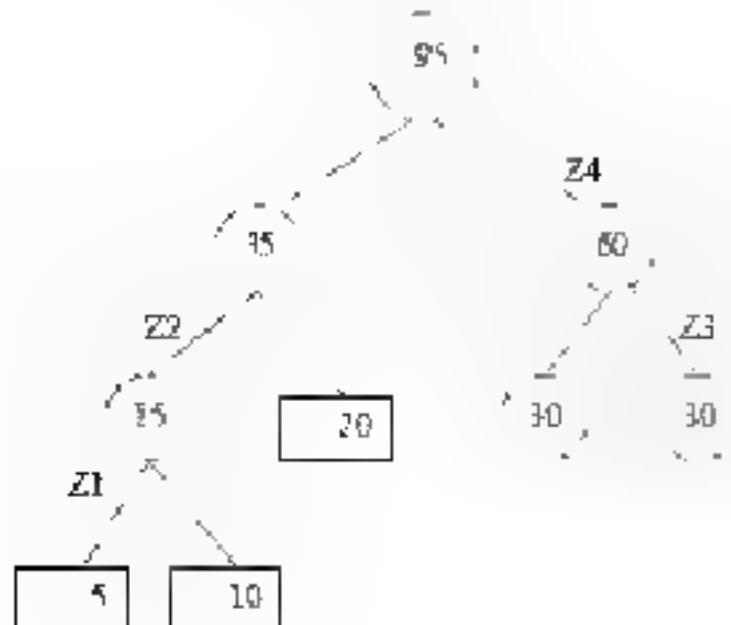
يوجد أدنى مجموعة ملفات مختارة بحيث يكون أدنى ملف واحد تحت وبقا عمليات وبقا تحريك تقود الختمة بملفات المدمجة أي أنه ستكون مسالة (K-Minimum).  
 أي هذه المسألة هي مسألة بحد (D) من الملفات الموزعة على شكل ابروج وبقا ملف واحد مرتب بكل عدد ممكن من الحركات السجلات ، أي هذه المسألة تستدعي ترتيب هذا من ابروج من الملفات المراد بحيث لتلك فهي لتعلق نموذج الترتيب

يراعده الموضح في

(التلخيص حركة السجلات الترخيص المتغير لأنك حجمه مع كل خطوة أولا).

مثال/ يجب مجموعة الملفات المرصحة كما في التالي.

5 في <10, 30, 10.5, 50> [F1, F5]



إن القيمة (20) تمثل عدد القنود في السجائب في السلك الأول، وهكذا بالتسوية يتبعه في  
القيمة (30) تمثل عدد القنود في السلك الثاني.  
في القيمة (10) تمثل طول السلك  
في ترتيب التفرع بعد التفرع هو الآخر.

$$20, 30, 15, 30 \\ 30, 30, 35 \\ 60, 35$$

إذا كانت  $W$  تمثل عدد السجائب الخارجيه  $W$  في  $W$  يمثل طول السلك  $W$   
في إلى بعد تكلي بحركات السجائب لشجرة التفرع التالية هذه تكون

$$\sum_{i=1}^n W_i \cdot W_i \\ = 5 \cdot 3 + 10 \cdot 3 + 20 \cdot 2 + 30 \cdot 2 + 30 \cdot 2 \\ = 205$$

وهذا هو الحد الأمثل الذي يكون هو الحجم الأمثل  
نعم، تطبيق عملية تفرع عشوائي.

وهذه هي الدالة التي تولد الشجرة للتفرع مكرية بـ C++.

```
Struct Tree node
{
    Struct tree* Lchild,*Rchild;
    Int Weight;
}
```

إن الجزء  $W$  خاص بشكل العدة للشجرة الشافية بالقيمة الموصولة

```
Typedef Struct Tree node type;
```

```
Type *tree (int n)
```

List is global list of n single node binary trees as described above.

$n$  تمثل عدد السلك  $W$  في  $W$  تمثل عدد القنود لكل سلك

0	W	0	.....	n
---	---	---	-------	---

```
For (int i=1, i<=n, i++)
```

```
{ type *pt=new type;
```

```
Pt->Lchild=Least (list);
```

```
Pt->Rchild=Least (list);
```

```
Pt->Weight=(Pt->Lchild->Weight)+(Pt->Rchild->Weight);
```

```
Insert (list, *pt);
```

```
}
```

إن هذا ال for يكون خاصه بتعبير الجمع بين السلك  $W$  بالدالة Least درجج مؤشر إلى  
الخطه التي تكون ذات  $W$  في السلك  $W$  بخطه مكافئ، الدالة insert تقوم بإضافة  
عنصر إلى القائمة list الخاصة بالعدد  $W$   $pt$  هو رقم محطرات المؤشر ،

```
Return (Least(list));
```

```
}
```

إن الناتج من هذه العملية هو مؤشر إلى شجرة التفرع التالية

و قمنا بالي توضيح موجد الخوارزمية

1. كل شجرة في القائمة List هناك دائما صيغة واحدة ، هذه الصيغة هي صيغة خارجية حيث تتكرر من ثلث جهات في (Rchild) ، (Weight) ، (Lchild).  
(Rchild) و (Lchild) متساويان هما متقابلة ، إذ أن كل (Weight) تحتوي على طرف بعد العلاقات المتساوية.  
2. الدالة (Tree) تتعامل دائما مع عنصرين هما (Least) و (Insert) حيث إن الدالة (Least) تقوم بإدخال شجرة في القائمة حتى إذا كانت لا تزال لا يوجد في القائمة فإشرف إلى هذه الشجرة ، وعوضا بذلك يحلها من القائمة.  
إذا الدالة (Insert) فإنها تقوم بإدخال شجرة تأتي حفرها Pt إلى القائمة (Least) حيث إن مؤشري هذه نقطة هي Pt.

3. في شجرة الترميز لتقليل التكلفة في جهة هذه الخوارزمية تستخدم أيضا أنه هناك سبب سببها حيث يتكرر الترميز على تلك السبب التي هناك للعنصر الأكبر في الشجرة.

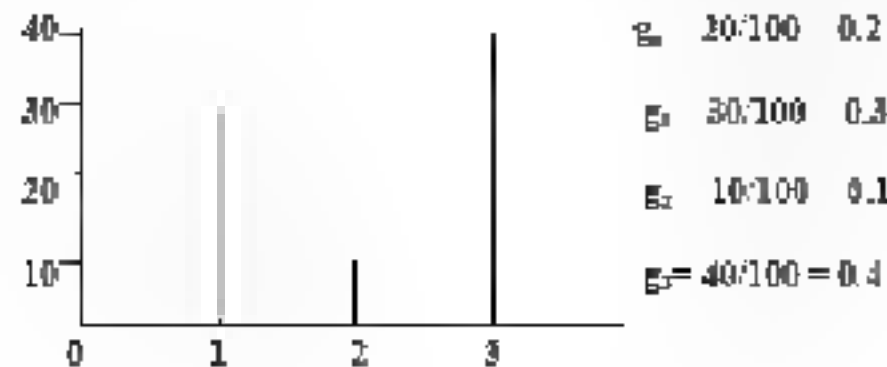
تعددت الوقت تخفيضه

إن صيغة for الرئيسية تتكرر (n-1) من المرة ، في حالة الاحتفاظ بالقائمة (List) من مرة تعدد عددا سبعة أي هم الأوزان في الحقول من القائمة (Least) تتكلف (O(1)) من الوقت والدالة (Insert) هناك لتجربتها بوقت هو (O(n)) أي إن الوقت الكلي المستغرق هو (O(n^2)).

الحل - مستخدم قائمة الترميز في بجدد لتسهيل البيانات -

طريقة هافمان (Huffman code) سبب سميت للعالم هافمان 1952 ، تعتمد على فكرة تقليل البيانات وتصميمها بدون التغير على خطة الترميز أي بدون فقدان البيانات

هافمان / ليف هافمان سلبية استخدم طريقة هافمان لتبسيط المجموع Greedy mile مسألة بالمرجع التكراري الثاني



مثال ضم المبرج



ب. ترتيب القيم وجميع القيمتين



ج. الاستمرار بجمع بقية الأصغر 5 أصغر الرموز بقيمتين فقط

مقود الاثنايية Original gray level (natural code)	الاحتمالية Probability	معدرة هرفمان Huffman code
$g_0 : 00_2$	0.2	$010_2$
$g_1 : 01_2$	0.3	$00_2$
$g_2 : 10_2$	0.2	$011_2$
$g_3 : 11_2$	0.4	$1_2$

يوجد Entropy الخاص بالاحتمالية المستخدمة

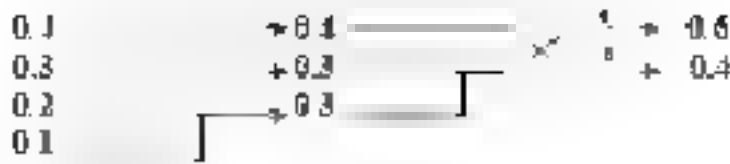
$$\begin{aligned} \text{Entropy} &= - \sum_{i=0}^3 P_i \log_2 (P_i) \\ &= [ (0.2) \log_2 (0.2) + (0.3) \log_2 (0.3) + (0.1) \log_2 (0.1) + \\ &\quad (0.4) \log_2 (0.4) ] = 1.846 \text{ bits / pixel} \end{aligned}$$

كما ان يوجد  $\log_2 (X)$  يمكن الحصول عليه حسب القانون التالي:  
 $\log_2 (X) = 1.322 * \log_{10} (X)$

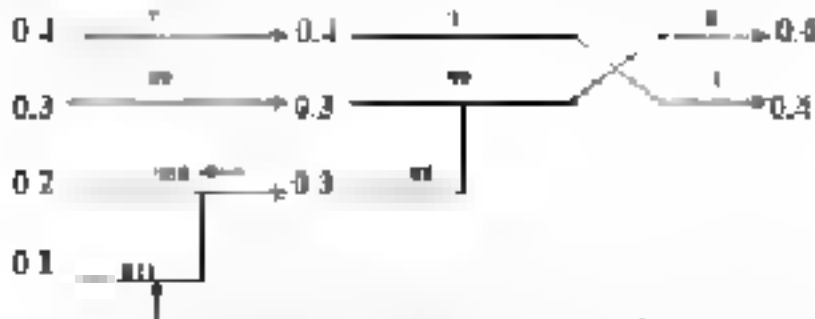
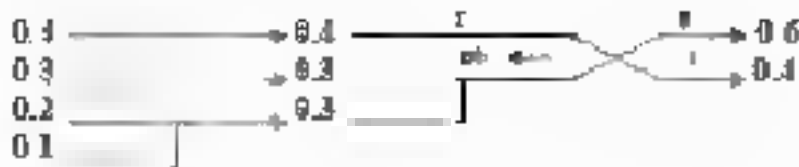
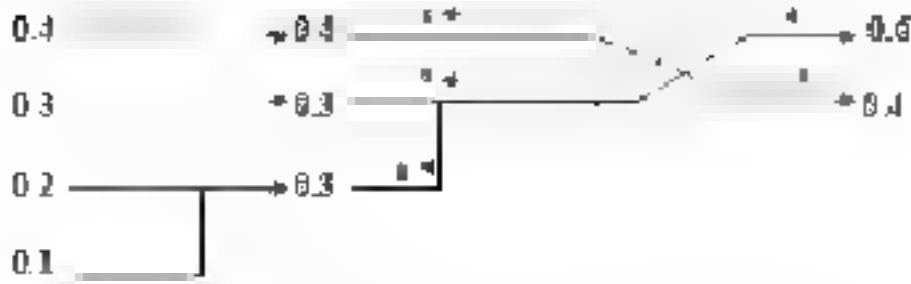
يوجد معدل طول الشفرة حسب القانون التالي (Average length)

$$\begin{aligned} \text{Lave} &= - \sum_{i=0}^3 L_i P_i \\ &= 3(0.2) + 2(0.3) + 3(0.1) + 1(0.4) = 1.8 \text{ bits / pixel} \end{aligned}$$

مطور التفرع



مطور غير حجب



مثال: لنفترض اننا نستخدم أمثلة لأحداث للجسج، ونحدد شعرة خروجه

Horizontal code Example (from our text)

Order	Probability	Order	Probability
1	0.4	1	0.4
2	0.3	2	0.3
3	0.2	3	0.2
4	0.1	4	0.1

Uniquely decodable			Not uniquely decodable			
Symbol	$P_i$	code	$s$	$t$	$u$	$v$
$a_1$	$\frac{1}{4}$	1	0.1	0.1	0.1	0.1
$a_2$	$\frac{1}{4}$	11	0.3	0.3	0.3	0.3
$a_3$	$\frac{1}{4}$	10	0.2	0.2	0.2	0.2
$a_4$	$\frac{1}{4}$	100	0.1	0.1	0.1	0.1
$a_5$	$\frac{1}{4}$	1000	0.0625	0.0625	0.0625	0.0625
$a_6$	$\frac{1}{4}$	10000	0.03125	0.03125	0.03125	0.03125

Although it might not look like it this is both uniquely decodable and instantaneous code  
 Think about how you'd decode an incoming bit stream  
Empirical test example

$$H = - (0.3 \log(0.3) + 0.3 \log(0.3) + 0.3 \log(0.3) + 0.1 \log(0.1) + 0.0625 \log(0.0625) + 0.03125 \log(0.03125))$$

$$H = 2.44$$

Average code length

$$E = 1(1) + 2(1) + 3(1) + 4(0.25) + 5(0.25) + 6(0.25)$$

$$E = 1.8$$

Note that  $H < E$  as expected, but we would be hard (impossible)  
 to find any code which did better (i.e. closer to the optimal  
 entropy  $H$ ). That is why the Huffman code is a 'compact' code

شجرة هوفمان مستخدمة لتشفير النصوص

الطريقة المستخدمة لتشفير كلاً من

- 1- نص، جعله نصية
- 2- حساب تكرار كل حرف في النص
- 3- عمل شجرة «نصف» بحيث لكل نصين في كل فرع
- 4- برقيده الشجرة بحيث كل مسار على اتجاه اليساري يعطى به (0) وكل مسار على اتجاه اليميني يعطى له (1).
- 5- كتابة شفرة حرفي لكل حرف في النص، خلال سبع خطوات
- 6- إيجاد معدل الترميز و Entropy

مثال، لووجد شفرة هوفمان للجملة التالية

dead beat safe decided dad. Dad faced a faced ab Dad a  
 caded \_ dad be back \_

Space	a	b	c	d	e	f	.
1"	12	4	5	10	12	4	4

## 2 جمع اول هفتین یکل عرب

Space	a	b	c	d	e	f	.
7	12	4	5	19	12	4	4

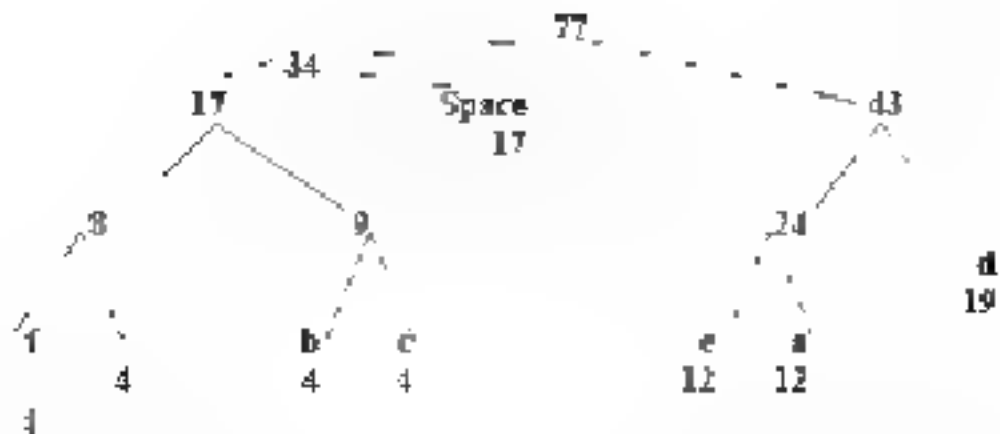
Space	a	b	c	d	e	f	.
17	12	4	5	19	12	4	4

Space	a	e	b	c	d	f	.
17	12	12	4	5	19	4	4

8	17	34	17	43	12
12	24	12			

8	17	34	77	43	19
12	24	12			

### 3. يرمز الشجرة



Space	a	b	c	d	e	f	.	رمز
01	101	0010	0011	11	101	0000	0101	شجرة هو

وحيث أن الخوارزمية الخاصة ببناء شجرة هو هافمان هي:

Huffman(C)

1.  $n = |C|$
2.  $Q = C$
3. for  $i = 1$  to  $(n - 1)$  do
4.     allocate a new node  $z$
5.      $z \text{ left} = x = Q \text{ Extract-Min}$
6.      $z \text{ right} = y = Q \text{ Extract-Min}$
7.      $f[z] = f[x] + f[y]$
8.     Insert( $Q, z$ )
9. ...return  $Q \text{ Extract-Min}$    return the root of the tree

حيث يمكن تقليل التعيد من  $n \log(n)$  إلى  $n^2$



الفصل الخامس  
البرمجة الديناميكية  
**Dynamic  
programming**

## 1.5 البرمجة الديناميكية (Dynamic programming)

سمّ نفعل البرمجة الديناميكية بطرق مختلفة نذكر في التالي نموذج برونس راسي  
آخر . وعموماً عن استخدام التتابع الفرعي و التكرار يصف نموذج البرمجة الديناميكية  
الإجراء الذي هو وجهة نظر "الحالات" والقرارات، والتحويلات والتفرعات وهي  
1- يبدأ بإجراء من حالة البداية حيث يتم اتخاذ قرار  
2- نسير في قرار وانتقل إلى حالة حتمية  
3- بالانتقال إلى الحالة التالية، لحظة النهاية والقرار 1. يتم الوصول إلى قيمة مرجعة  
4- نسير الإجراء غير مضمّن من الحالات حتى الوصول إلى حالة معينة

تكون مشكلة في إيجاد الحسنة التي تحصل القيمة تكلفة البرمجة 'محمّد' وتعد نموذج  
وسايل البرمجة الديناميكية الأكثر ملاءمة للحلّ الذي يمر من تسلسل معين يستمر في  
البرمجة للبرمجة التي تقع في هذا التكرار، عندما تكون مجموعة القرارات "محدودة" ومقطعة  
وعندما يكون التتابع الفرعي خطي.  
وقد وصف البرمجة الديناميكية بأنها الطريقة الإجمالية طرق الأمثلة مستمرة على حل  
صعب واسع من المشاكل، يبين هذا النموذج مشاكل معينة بشكل خاص حيث يمر بسير إلى  
الحالات حسابه فعلاً، في تلك الحالات التي تتضمن توازن غير مستقر، فيم مقطعة وهي  
هذه الحالات، قد تشكل البرمجة الديناميكية سهلية لنحلّ الوحيد للمشكلة

أو يعرف بين فائده فجموع (Greedy method) التي تعتمد على خطوات واضحة ووض  
البرمجة الديناميكية (dynamic programming) التي تعتمد على معلومات علمية في  
البرمجة يتم بتعاقب قرارات واحد يتوقف النتيجة بوجود تعاقب قرارات كثيرة تكون القرارات التي  
تتكون قرارات جزئية، غير مثلى لا يمكن أن تكون مثلى ولهذا لا تأخذ

فإن كان لدينا D من القرارات لكل قرار من هذه D' تعاقب قرار ممكن  
لأنه حوار وميف البرمجة الديناميكية لها بتقيد مسحة الحزم

### 3.1 مثله على البرمجة الديناميكية

مثال // إيجاد الوقت الأمثل لـ n من القيم كالتالي

- $a^i$  dominates  $a^j$  if  $a^i > a^j$  since

$$\lim_{n \rightarrow \infty} a^i = 0 \quad \lim_{n \rightarrow \infty} a^j = 0$$

- $a^i + a^j$  dominates  $a^k$  since

$$\lim_{n \rightarrow \infty} (a^i + a^j) = 0 \quad \lim_{n \rightarrow \infty} a^k = 0$$

Complexity	10	20	30	40
$n$	0.00001 sec	0.00002 sec	0.00003 sec	0.00004 sec
$n^2$	0.0001 sec	0.0004 sec	0.0009 sec	0.0016 sec
$n^3$	0.1 sec	0.8 sec	2.7 sec	6.4 sec
$n^4$	1 sec	16 sec	243 sec	256 sec
$n^5$	101 sec	320 sec	2430 sec	10240 sec
$2^n$	0.02 sec	10 min	20 years	256000000 sec

مثال: ليوضح حور فييه إيجاد أقصر مسافة

What is  $d[i, j]^m$ ?

$$d[i, j]^0 = 0 \text{ if } i = j$$

$$= \infty \text{ if } i \neq j$$

What if we know  $d[i, j]^{m-1}$  for all  $i, j$ ?

$$d[i, j]^m = \min(d[i, j]^{m-1}, \min_k (d[i, k]^{m-1} + w[k, j]))$$

$$= \min(d[i, k]^{m-1} + w[k, j]) \quad 1 \leq k \leq n$$

since  $w[k, k] = 0$

This gives us a recurrence with which we can evaluate  $d$  in a bottom up fashion

```

for i = 1 to n
  for j = 1 to n
     $d[i, j]^m = \infty$ 
    for k = 1 to n
       $d[i, j]^0 = \text{Min}(d[i, k]^m, d[i, k]^{m-1} + d[k, j])$ 

```

This is a  $O(n^3)$  algorithm just like that x shortest path problem but it only goes from  $m$  to  $m+1$  edges

مثلاً، لحدة قلعة الوقت قلعة هورجو على أن يتم تحديد كما هو موضح في الجدول التالي:

n	$f(n) =$	$f(n) = n^2$	$f(n) = 2^n$	$f(n) = n^n$
10	0.01 $\mu s$	0.01 $\mu s$	10 $\mu s$	10.63 ms
20	0.02 $\mu s$	0.4 $\mu s$	1 ms	17.4 ms
30	0.03 $\mu s$	0.9 $\mu s$	1 sec	$8.4 \times 10^4$ years
40	0.04 $\mu s$	1.6 $\mu s$	$1.8 \times 10^6$	
50	0.05 $\mu s$	2.5 $\mu s$	13 days	
100	0.1 $\mu s$	10 $\mu s$	$4 \times 10^{30}$ years	
$1 \times 10^4$	1.00 $\mu s$	1 ms		

### 3-5. تجميع البيانات (Data clustering):

تجميع البيانات هي عملية وضع البيانات في مجموعات مختلفة، خوارزمية التجميع تقوم بمجموعة من البيانات إلى عدة مجموعات، حيث أن التجميع هو عملية تجميع البيانات من التشابه بين بعض من مجموعات. فالتجميع هو عملية تجميع البيانات في فئات بسيطة على مبعدها وهي ثنائية جد من أجل أن يكون في طريقة التفكير حيث أن كل مجموعة مع كمية كبيرة من البيانات يجب أن يتم التجميع إلى فئات الكمية لتسهيل من التجميع إلى عدة فئات من المجموعات والعلاقات، وبذلك من سحر تجميع البيانات التجميع.

خوارزمية التجميع تستخدم على نطاق واسع بين فقط للتطبيق وتحسين البيانات وإتاف هي طريقة لتجميع البيانات بناءً على قرب البيانات، حيث أنه في كل مجموعة من البيانات وفي البيانات وفي الإمكان بناء نموذج للمشكلة على أساس تلك المجموعات. هناك عدد من التقنيات المستخدمة في عملية تجميع البيانات، ومن هذه التقنيات (الخوارزمية) التي سوف يتم الحديث عنها بشكل مفصل.

- K-means Clustering
- Subtractive Clustering

### 3. خوارزمية الـ (K-means Clustering)

هي خوارزمية تجمع عند من البيانات لتتخذ إلى خصائص ومميزات هذه البيانات، ويتم عليه التجميع من خلال بعض المسافات بين البيانات ومركز التجميع (centered cluster). وتتم هذه الخوارزمية من خلال الخطوات التالية:

1. حساب المسافات بين كل البيانات ومركز التجميع.
2. حساب المسافات بين كل البيانات ومركز التجميع.
3. تجميع البيانات وتصنيفها في مجموعات بناءً على المسافات بين المركز ونقاط البيانات.
4. اعتماد بعض الخطوات من 1 - 3 حتى الوصول إلى حالة التثبيت.

يعتمد لدى هذه الحوزة على المراكز الأولية لمراكز التجمع (Centered)، ومن المستحسن تنفيذ هذه الحوزة بعبء غير ١. هي اختلاف المراكز في كل مرة عن المراكز السابقة.

تفرض لدى أربعة أنواع من الأدوية، وكل نوع من الأدوية لديه عدد من المراكز في هذا المجال كل نوع من المراكز.

نوع الدواء (Medicine)	مؤشر الورق (Vaght Index)	أعداد المراكز (PH)
A	1	1
B	2	1
C	4	3
D	5	4

الهدف من هذا المثال هو جمع أنواع من الأدوية في مجموعتين اعتماداً على سعة كل نوع من الأدوية، وتحقيق هذا الهدف على تنفيذ خطوات خوارزمية التجميع كالتالي:

١. التقييم الإحصائي لمراكز التجمع.  
تفرض أن الدواء A والدواء B هما في كل التجمع الأولى، لكن A و C على اختلاف المراكز حيث أن (A, B) و (A, C) و (B, C) يبين الشكل توزيع أنواع الأدوية المعبر عنه بالمعيار (A, B) و (A, C) و (B, C) يبين مراكز التجمع الإحصائية مع المراكز بعينها، أي هذه المراكز قد نفساً بشكل عشوائي.

٢. المسافات بين النقاط والمراكز.  
نحسب المسافة بين مركز التجمع وكل نقطة من النقاط في المستوى لننتج عدد مصغره من المسافات، حيث في كل مستوى في المستوى المسافات. ننتج نوع دواء من حيث المسافة من مصغره المسافات. يتكون من المسافات بين كل نقطة ومركز التجمع الأول، والمسافة التي يتكون من المسافات بين كل نقطة ومركز التجمع الثاني.

٣. جميع النقاط.  
حيث نحصل كل نقطة بين مركز التجمع إلى المسافة على كل مسافة، وهكذا فإن الدواء (A) يذهب إلى المجموعة الأولى، الدواء (B) إلى المجموعة الثانية، الدواء (C) إلى المجموعة الثالثة، والدواء (D) يذهب إلى المجموعة الرابعة.  
يخرج لدينا مصغره المجموع على G، التي تتكون من القيم 1 و 0، ويكون المعنى في مصغره المجموع على 1 فقط إذا كان الدواء مسدداً إلى تلك المجموعة.

٤. التكرار الأول: تحديد مراكز التجمع.  
بعد مرحلة العناصر كذا المظهر، نحسب مركز جديد لكل مجموعة اعتماداً على هذه المراكز الجديدة. المظهر الأولي يتكون من عناصر واحد فقط ونفساً إحصائياً لمركز التجمع الأول، كما هي دور تغيير (A, B) و (A, C) أو بالمجموعة الثانية والتي ستكون من قادم العناصر بتغيير إحصائياً مركز التجمع الثاني بناءً على إحصائيات العناصر الخاصة.

5- التكرار الأول: المسافات بين النقاط والمراكز  
في هذه الخطوة يتم حساب المسافة بين كل نقطة ومركز التجمع الجديد، كما في الخطوة  
التالية، ينتج لدينا مصفوفة من المسافات

6 التكرار الأول: جميع النقاط  
على مركز الخطوة التالية: نحيل كل نقطة إلى مركز تجمع بالإعتماد على أقل مسافة بالمجموعة  
في مصفوفة المسافات المحددة بنقل النوى التالي (B) إلى المجموعة الأولى، ننقل باقي  
النوى كما هي فنظهر مصفوفة المجموعات

7 التكرار التالي: تحديد مركز التجمع  
الآن نمرر بمجموعة الخطوة الرابعة بحساب تحديد مركز التجمع الجديد بالإعتماد على  
عنايه التجميع في التكرار الأول حيث تكون كل من المجموعة الأولى والثانية من عناصره

8 التكرار التالي: المسافات بين النقاط والمراكز  
نكرر الخطوة التالية، ينتج لدينا مصفوفة مسافات جديدة

9 التكرار التالي: قصص النقاط  
نمرر بنوى نحيل كل نقطة إلى مركز تجمع بالإعتماد على أقل مسافة

نتبع لدينا في النهاية معادلة التجميع بين التكرار الأول والتكرار الثاني، نلاحظ من  
المجموعة يتم تغير من حيث عناصرها وهذا يعني أن عملية الحساب في الـ (k-mean  
clustering) وصلت إلى حالة الثبات، وهذا يعني أن هذه الخوارزمية لم تعد بحاجة إلى تكرار  
من التكرار، وبالتالي حصلنا على النتيجة النهائية للتجميع

## 2 خوارزمية 2 (Subtractive Clustering)

المشكلة في طريقة التجميع السابقة (Mountain Clustering) هي أن الحساب الحسابية  
يرتفع طرديا مع عدد النقاط في المشكلة، وذلك لأنه كلما تكرر سابقا قدم تقسيم الـ Mountain  
Function عند كل نقطة تقاطع في الشبكة على محورتي الـ (x, y)  
استخدمت خوارزمية الـ (Subtractive Clustering) حل هذه المشكلة وذلك بتوزيع عدد  
من نقاط البيانات لتكون مراكز للمجموعات، بدلاً من استخدام نقاط تقاطع خطوط الشبكة كما هو  
الحال في الـ (h/c)، وهذا يعني أن التعديل الحسابية أصبحت تتناسب مع حجم المشكلة بدلاً من  
المعادلة

خوارزمية الـ (Subtractive clustering) هي عملية تحديد مركز التجمع والمجموعات التي  
تتبعها، صفة المشكلة من كل أن عدد نوى لتعلم بعد المجموعات المقترحة لتبدأ  
وبعد هذه الطريقة على حساب كثافة البيانات عند كل نقطة ضمن مستوى معين، لهذا كثافة  
كل نقطة مرشحة لتكون مركز تجمع، فكله يمكن إيجز كثافة البيانات عند نقطة x من المعادلة  
التالية

حيث أن  $\pi_a$  ثابت يجب أن يكون ثابتاً حول كل نقطة يتم حساب الكثافة عنده هذه المعادلة  
وكما نرى هذا القطر أصبح لدينا عدد اثنين من المجموعات، وكلما قل القطر زاد عدد  
المجموعات، وبما أن نوى قيمة  $\rho_b$  أكبر من قيمة  $\pi_a$  غالباً يستخدم  $\rho_b = 1$ ، وذلك لتقليل  
تأثير الكثافة عند نقاط المجاورة لنقطة المركز الأولى

تم حساب المراكز الأولى 100 والتي كانت كثافة الحساب عنده أعلى مما يمكن (Dij). بعد ذلك يتم حساب قيم الكثافة لجميع العقد كل نقطة.

- وتقوم خوارزمية الـ (Subtractive clustering) بالخطوات التالية
1. إيجاد نقطة معينة موجودة في المجال تكون عدد من الكثافة عالية ويتم حساب الكثافة من المعادلة الأولى. وهر تم اختيار نقطة معينة كمركز ، وبذلك عن طريق وجودها بين عند كثير من النقاط المحيطة.
2. يتم حذف نقاط النجاس
3. يتم تبحث الخوارزمية عن مركز جديد وذلك عن طريق حساب قيم الكثافة للنقاط الأخرى كما في المعادلة الثانية ، وتستقر هذه العملية حتى لا يتغير من كل النقاط أو يوجد عدد كافٍ (مستوى) من المعجم.
- بعد إجراء هذين الخطوات خوارزمية هي لها تكرار لعملية من الخوارزمية حيث يتم تكرارها حتى
- كما نرى الأمر في شكل المخطط التالي

#### 4-5 خوارزمية (Dijkstra)

يخترع ديكسترا (Edsger Dijkstra) هو أحد العلماء الهولنديين في علوم الحاسب ، ولد في 1919 في مدينة روتردام ، ولد مشواره العلمي بمجال نظريات الخوارزمية في جامعة ليون. كان من عمل ما تركه في الخوارزمية منصب في علوم الحاسب

استلم ديكسترا عام 1972م جائزة ACM Turing على تطوير مساهمته الأساسية في برمجة الخوارزمية كما حظي بمناصبه في (كرسي ستيفن جوردن في نظرية الخوارزمية) في جامعة تكساس في بوسطن منذ عام 1984م وحتى تقاعد عام 2000م

من أبرز إسهاماته في علوم الحاسب هي خوارزمية الخوارزمية لأقصر المسارات والتي تعالج أيضاً بحوارزمية ديكنستر ، مستخدم هذه الخوارزمية في تنظيم نقل المعلومات بين أجهزة الحاسب وعرف كيف بعد خوارزمية الطرق لأقصر المسارات

كتب ديكسترا عام 1958م ورقة بحثية هامة في تخصص علوم الحاسب التي تسمى الخوارزمية الخوارزمية للعثور على المسار الأقصر ، عبارة البرمجة الشهيرة (ثاني، ثم أكثر استخداماً "more use a for 2") (والتي تعبر إلى حقيقة أنه يجب تجنب تعقيد تقدم أكثر من عمل خلية معينة لأنه حتى يوجد شخص هذا المصطلح داخل هذه الخوارزمية

والخوارزمية الخاصة بهذه الطريقة هي

DIJKSTRA( $G, s$ ).

1. INITIALIZE\_SINGLE-SOURCE( $G, s$ )
2.  $S \leftarrow \{ \}$   $S$  will ultimately contain vertices of final shortest-path weights from  $s$
3. Initialize priority queue  $Q$  i.e.  $Q \leftarrow V[G]$
4. while priority queue  $Q$  is not empty do
5.  $u \leftarrow \text{EXTRACT\_MIN}(Q)$   $\therefore$  Pull out new vertex

```

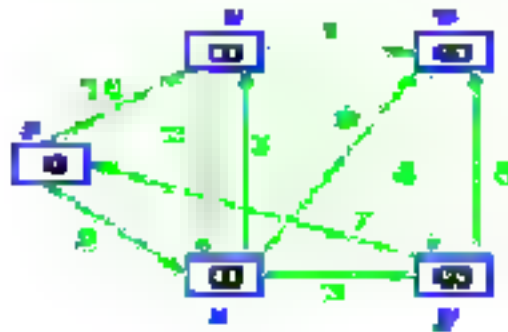
6   S ← S ∪ {u}
      Perform relaxation for each vertex v
      adjacent to u
7   for each vertex v in Adj[u], do
8     Relax (u, v, w)

```

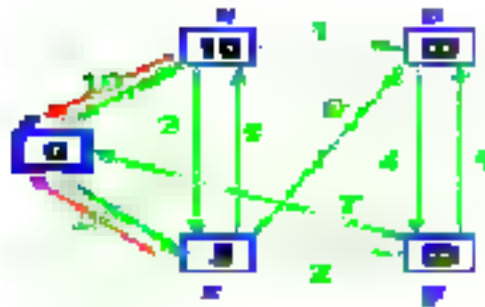
### 5.5 إنشاء تطبيق خوارزمية (Dijkstra)

مثال // تطبيق الخوارزمية كما في الخطوات التالية

1 الاسم مستطى  $G=(V, E)$  كل قبة هناك كات عبر سوية عا العدة  $S$  حيث كات  $Q$

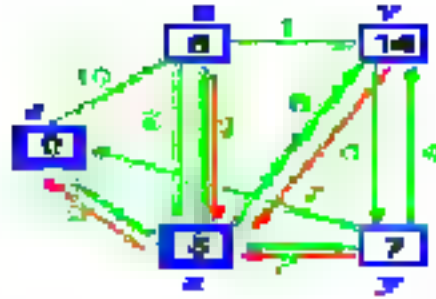


2 أولا نحدد عقدة أوله من  $S$  ونحدد  $d[S]$

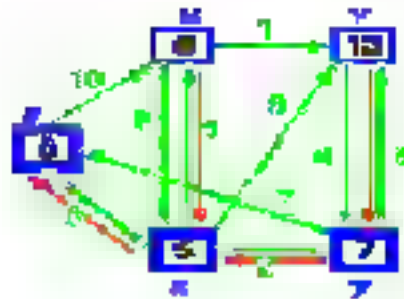


3 الأخير عدة  $x$  ونطبق خطوات الخوارزمية

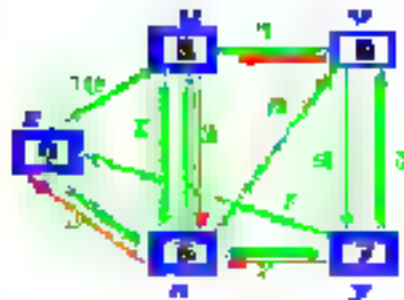




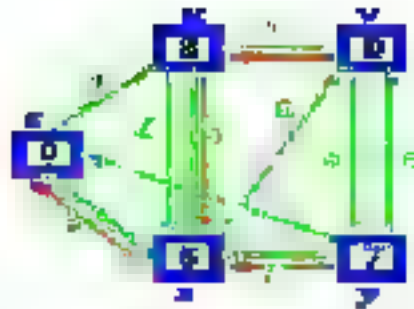
4. اكتب  $\psi$  قيمة البعد  $\psi$  وبتطبيق هذه الخطوات



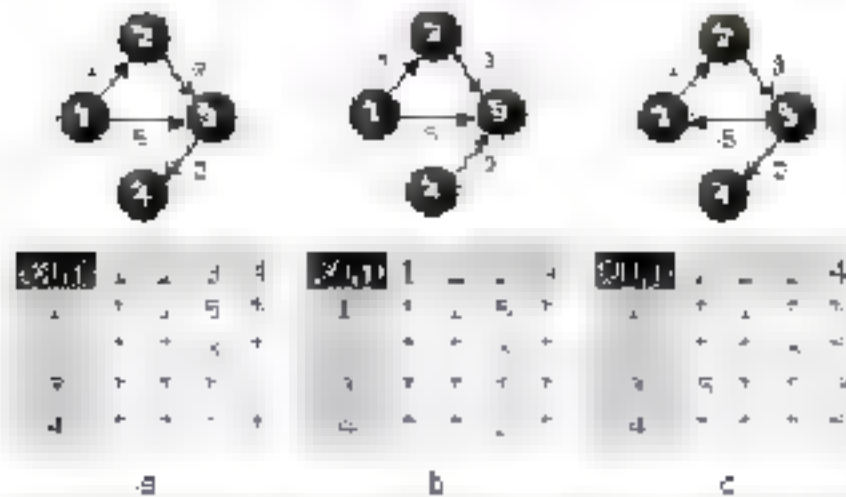
5. الآن هناك المقام  $\psi$  نحرك عقدة  $\psi$  جاريها  $V$



6. عقدة  $\psi$  لا تقطع سوف يعطى أصغر مسار  $\psi$  غير بصيف لعقدة



مثلاً • نقوم بتطبيق الطريقة المقترحة ليترك لنا المخطط الآتي



let  $C = \{1, 2, \dots, n\}$  denote the set of cities and for each city  $j$  in  $C$  let  $P(j)$  denote the set of its immediate predecessors and let  $S(j)$  denote the set of its immediate successors, namely set

$$P(j) = \{k \in C \mid D(k,j) < \text{Infinity}\}, j \in C$$

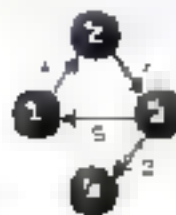
$$S(j) = \{k \in C \mid D(j,k) < \text{Infinity}\}, j \in C$$

Thus, for the problem depicted in Figure 1 a)  $P(3) = \{0\}$ ,  $P(2) = \{0\}$ ,  $P(4) = \{0, 1\}$ ,  $P(5) = \{0, 1, 2\}$ ,  $S(0) = \{1, 2, 3\}$ ,  $S(1) = \{4, 5\}$ ,  $S(2) = \{5\}$ ,  $S(3) = \{6\}$ ,  $S(4) = \{6\}$ ,  $S(5) = \{6, 7\}$  where  $\{\}$  denotes the empty set

Exercise 18. If  $\mathcal{D}$  is the set of all  $\mathcal{D}$  on  $\mathcal{D}$  with no immediate predecessor and for  $\mathcal{D}$  is the set of nodes that have a immediate predecessor. Then  $\mathcal{D}$  is a poset.

$$\mathcal{D} = \{1, 2, 3, 4, 5\}$$

$$\mathcal{D} = \{1, 2, 3, 4, 5\}$$



$\mathcal{D}(i, j)$	1	2	3	4
1	*	1	*	*
2	*	*	3	*
3	-5	*	*	2
4	*	*	*	*

a



$\mathcal{D}(i, j)$	1	2	3	4	5
1	*	0	*	*	*
2	*	*	1	*	4
3	*	*	*	3	4
4	*	5	*	*	2
5	*	*	*	*	*

b

Matrix	1	2	3	4	5	6	7	8	9	10
1	*	1	*	*	*	*	*	*	*	*
2	*	*	3	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*	*	*	*
4	*	*	*	*	*	*	*	*	*	*
5	*	*	*	*	*	*	*	*	*	*
6	*	*	*	*	*	*	*	*	*	*
7	*	*	*	*	*	*	*	*	*	*
8	*	*	*	*	*	*	*	*	*	*
9	*	*	*	*	*	*	*	*	*	*
10	*	*	*	*	*	*	*	*	*	*

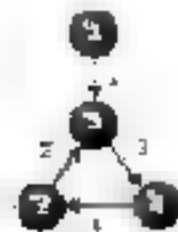
يتم هذا إيجاد الصور المتساوية

$x_{ij}$  Quantity (Flow) sent along the link from city  $i$  to city  $j$   
 $i=1, \dots, n$

then the minimum cost network flow problem is as follows

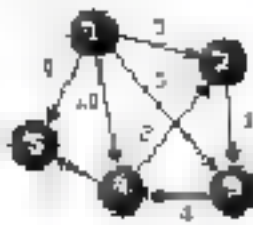
$$\begin{aligned} \min \sum_{i,j} f_{ij} x_{ij} \\ \text{s.t.} \quad \sum_{j=1}^n x_{ij} - \sum_{k=1}^n x_{ki} = a_i \quad i=1, \dots, n \end{aligned}$$

مثال: توضيح الطريقة



$$\begin{aligned} f_{12} &= 0 \\ f_{13} &= 4 + f_{43} \\ f_{24} &= \min \{2 + f_{42}, 1 + f_{11}\} \\ f_{34} &= 3 + f_{43} \end{aligned}$$

مخطط



$f_{ij}$	2	3	4	5
1	4	3	10	9
2	5	2	1	4
3	4	2	4	1
4	4	4	4	4

a

$f_{ij}$	2	3	4	5
1	4	3	10	9
2	5	2	1	4
3	4	2	4	1
4	4	4	4	4

b

الص

Iteration k	J	F	Iteration k	U	F
0	$\{1,2,3,4,5\}$	$(0, \infty, \infty, \infty)$	0	$\{1,2,3,4,5\}$	$(\infty, \infty, \infty, \infty)$
1	$\{2,3,4,5\}$	$(1, 5, \infty, \infty)$	1	$\{2,3,4,5\}$	$(1, 5, \infty, \infty)$
2	$\{3,4,5\}$	$(2, 5, 2, \infty)$	2	$\{3,4,5\}$	$(1, 5, 2, \infty)$
3	$\{4,5\}$	$(0, 5, 3, 7, 9)$	3	$\{4,5\}$	$(0, 5, 3, 7, 9)$
4	$\{5\}$	$(0, 5, 3, 2, 8)$			

$$F_{k+1} = F(5) = 9 \times F(4) = F(5) = 7.$$

الخوارزمية لتابعه العتال تكون كالآتي

Initialize  $k=1$   $F(1) = 0$   $F(\infty) = \text{Infinity}$   $j = \infty$   $n$   
 $U = \{1, 2, 3, \dots, n\}$   
 Iterate While  $(U) \neq \emptyset$  and  $F(j) < \text{Infinity}$  Do  
 $U = U - \{j\}$   
 $F = \min_{i \in U} \{F(i) + D(i, j) + F(j+1)\}$   $j = \arg \min_{i \in U} \{F(i) + D(i, j) + F(j+1)\}$   
 End Do

Range	Sparsity	Acyclic	$D(i,j) \leq 0$	Gen					
File	Start at	Generated new problem	New algo	Solve Help					
N=10	?	?	?	?	?	?	?	?	?
DIP(i,j)	?	?					?	?	?
D(i,j)	?	?		?	?	?	?	?	?
D(i,j)	?	?							
			4	5	6	7	8	9	10
1	6		6	10	7	8	5	3	1
2		3	5	?	7	0	7	9	
3			2	6	3	0	4	5	
4				4	7	0	4	7	
5					10	5		3	5
6							4	6	8
7							6	9	
8								9	
9									8
10									

هذه هي الطريقة المستخدمة في حركة الأربوب ويجب أن تكون متساوية

Example arena:

```

X-----
B   B
B-----
B-----
BB-----
--B-----
      B.B
--B-B-----
B----- Y

```

Input

```

0 0
. 0
. 1
. 6
2 0
3 2
4 4
4 5
5 4
6 5
6 7
7 0

```

Sample Output:

```

0 0
0 1
0 2
. 2
2 2
2 3
2 4
2 5
2 6
3 6
4 6
5 6
6 6
7 6
7 7
. 1

```

## PROGRAM ( TRIED AND TESTED IN TURBO C++

```
#include <stdio.h>
#include <string.h>
struct Matrix { short int **array,
                int row,int col;
                };
struct Vertex { int num; int curDist
                };
typedef struct Matrix matrix;
typedef struct Vertex vertex;
void getGrid(matrix &m)
void getShortestPath() /* Dijkstra Algorithm */
void printSolution,int p[],int index);
matrix m;
int main()
{
    getGrid(m)
    getShortestPath()
    printf("n 1 1'm a");
    free(m.array);
    return 0;
}
void getGrid(matrix &m)
{int ctr1,ctr2 blockedSquares,x,y
scanf("%d%d%d%d",&m.row,&m.col,&blockedSquares)
m.array=(short int **)malloc(m.row*sizeof(short int *)),
for(ctr1=0;ctr1<m.row;ctr1++)
m.array[ctr1]=(short int *)malloc(m.col*sizeof(short int));
for(ctr1=0;ctr1<m.row;ctr1++)
for(ctr2=0;ctr2<m.col;ctr2++)
m.array[ctr1][ctr2]=0;
for(ctr2=0;ctr2<blockedSquares;ctr2++)
{
    scanf("%d%d",&x,&y);
    m.array[x][y]= 1
}
}
```

```

void getShortestPath() /* Uses Dijkstra Algorithm */
{
    int ctrl = 0, ctr2 = 0, row1, col1, row2, col2,
    int *predecessor = (int *)malloc(sizeof(int) * mrow * mcol);
    vertex *toBeChecked, minVertex,
    toBeChecked = (vertex *)malloc(sizeof(vertex) * (mrow * mcol + 1));
    for (ctrl = 1; ctrl <= mrow * mcol; ctrl++)
        predecessor[ctrl] = 3.000;
        toBeChecked[ctrl].num = ctrl - 1,
        toBeChecked[ctrl].currDist = 3.000;

    predecessor[0] = 0;
    toBeChecked[0].num = toBeChecked[0].currDist = mrow * mcol;
    toBeChecked[1].currDist = 0;
    while (toBeChecked[0].num != 0)
    {
        minVertex = toBeChecked[1], ctr2 = 1;
        for (ctrl = 1; ctrl <= toBeChecked[0].num; ctrl++)

            if (toBeChecked[ctrl].currDist < minVertex.currDist)
                {ctr2 = ctrl; minVertex = toBeChecked[ctrl]}
            },
        },
        toBeChecked[ctr2] = toBeChecked[toBeChecked[0].num];
        toBeChecked[0].num =
        row1 = minVertex.num / mcol; col1 = minVertex.num % mcol;
        for (ctrl = 1; ctrl <= toBeChecked[0].num; ctrl++)

            row2 = toBeChecked[ctrl].num / mcol;
            col2 = toBeChecked[ctrl].num % mcol;
            if (marray[row2][col2] == 0)
            if (((col1 < col2) * (col1 < col2) == 1 && row1 == row2) || ((row1
                row2) * (row1 - row2) == 1 && col1 == col2))
            if (toBeChecked[ctrl].currDist > minVertex.currDist + 1)

                toBeChecked[ctrl].currDist = minVertex.currDist + 1,
                if (toBeChecked[ctrl].num == 0)
                predecessor[toBeChecked[ctrl].num] = minVertex.num;
            }
        },
    }
}

```



```

printSolution(predecessor m,row*m.col-1);
}
void printSolution(int p[],int index)
{
    if(index==0)
        printf("0 0");
        return;
    };
    f(p[index]==31000)
        return;
    printSolution(p,p[index]);
    printf("%d %d",index/m.col,index%m.col);
}

```

## 6-5 المسحوظ متعدد المراحل (Multistage graph)

هو مسطح موجبة-قيمة تقسم بعدد إلى  $(K \geq 2)$  المجموعات منفصلة  $(V_1)$  حيث  $(1 \leq i \leq K)$  المجموعة  $(V_i)$  تتكون بحيث  $(|V_i| \geq 1)$  (أي أنه عند العناصر الموجودة  $(i=1)$  حيث  $i$  المجموعة الأولى والثانية تحتوي على عقد واحد

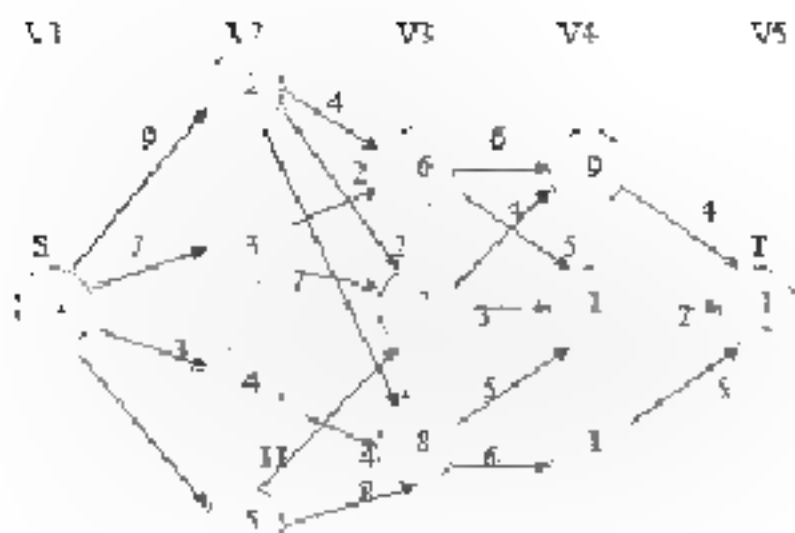
الفر من  $S$  إلى  $t$  في عقدة البداية في  $V_1$  وإلى  $t$  في عقدة النهاية في  $V_K$  والفر من  $V_i$  إلى  $V_{i+1}$  تكون كلفة الحافة بين  $(i, r)$  كما في المعادلة التالية

$$C(r, i) = \min_{r \in V_{i+1}} \{C(r, i) + \cos i(r+1, i)\}$$

كف يرمز للحافة الموجودة بكل مسطح  $(i, r)$  (د.ج.ب.)

إلى كلفة المسار من البداية  $S$  إلى النهاية  $T$  في مجموع كلف الحواف على المسار

ملاحظة 1: مساهم المسطح متعدد المراحل في إيجاد مسار أقل كلفة من  $(S$  إلى  $T)$  كل مجموعة  $V_i$  تفصل من نقطة في المسطح وكل مسار من  $(S$  إلى  $T)$  يبدأ (بالمرحلة 1) وينتهي (بالمرحلة  $K$ ).



شكل رقم (24) يوضح مخطط التدفق العرسل

صياغة البرمجة التفاضلية بحسب مخطط  $I_k$  من مراحل

6.3. الطريقة التصاعدي (Forward approach)

- (i) نلاحظ على كل مسار من  $S$  إلى  $T$  يكون نتيجة تعاقب  $(k-2)$  قرار ، حيث  $k$  قرار (i) لنفرض نحدد به عقدة  $(V_k + 1)$  حيث  $(2 \leq k \leq K)$  تكون على العنبر  $P_{k-1}$  هو العنبر الذي كان كلفه من العقدة (i) في العنبر  $P_{k-2}$  إلى العقدة (ii) يعطى إن  $P(2,2)$  حيث (i) يعقل عقده (ii) يعقل مرحلته من السراجل والى (iii) يعقل كلفه تلك العنبر يعطى  $Cost(2,2)$  وبمستعمل الطريقة التصاعديه (أي أنها تطلق قسماً) يعطى ذلك إننا تبدأ من الأخير (العقدة T) إلى الأولى (العقدة S).

$$Cost(i, r) = \min_{r \in J_i + 1 \leq j \leq E} \max \{c[i, j, r] + \cos t(j+1, r)\} \quad 1$$

وهي كلفة المسار الأقل كلفة حيث

في ذلك  $\langle r, r \rangle$  هي حكة تنتمي إلى جميع الحواف الموجودة في المخطط بحيث

$$Cost(k+1, j) = \begin{cases} c[j, r] & \text{if } j < j+1 \text{ and } R \\ = & \text{if } j < j+1 \text{ and } R \end{cases}$$

ولهذا فإن المعقنة رقم (1) تقطع للحالة  $Cost(2,5)$  بصفاً إلى

$$Cost(k+2, r), \forall j \in F_{k+1}$$

شريطة ذلك

$$Cost(k+3, j), \forall j \in F_{k+2}$$

وهكذا نسير على أن نصنع في كل خطوة موجودة في  $P(1)$  وبخلاف  $Cost(2,5)$  حيث نجد  
خيار قيمة للمسار الأقل كلفة

ويعتبر المخطط الموجود في الشكل رقم (2) السابق متصل على

$$Cost(3,6) = \min\{5 + \cos t(4,9), 5 + \cos t(4,10)\} = 7$$

وهي قيمة المسار الأقل كلفة

$$Cost(3,7) = \min\{4 + \cos t(4,9), 3 + \cos t(4,10)\} = 7$$

$$Cost(3,8) = \min\{5 + \cos t(4,10), 6 + \cos t(4,11)\} = 7$$

$$Cost(2,2) = \min\{4 + \cos t(3,6), 2 + \cos t(3,7), 1 + \cos t(3,8)\} = 1$$

$$Cost(2,3) = \min\{2 + \cos t(3,6), 7 + \cos t(3,7)\} = 9$$

$$Cost(2,4) = \min\{11 + \cos t(3,8)\} = 18$$

$$Cost(2,5) = \min\{11 + \cos t(3,7), 8 + \cos t(3,8)\} = 15$$

$$Cost(1,1) = \min\{9 + \cos t(2,2), 7 + \cos t(2,3), 3 + \cos t(2,4), 2 + \cos t(2,5)\} = 6$$

لاحظ أن الحد الأدنى لصورة المسألة كانت أنه الأحمر حيث أن هذا كلفة المسار تم  
صياغتها في كلفة  $Cost(1,1)$  وتعني كلفة  $Cost(1,5)$

ويبدأ من المسار الأقل كلفة هو  $S$  إلى  $T$ ، كان بأكمله متعارفاً 16 واختياد المسار بسجل  
المراتب المستخدمة في كل حالة (عند)

ونفترض أن  $D[i]$  هي قيمة  $T$  التي تصغر العلاقة التي ذكرناها سابقاً وهي

$$c[i, r] + \text{Cost}(\{i + 1, r\})$$

ملاحظة: //  $D[i, r]$  يمثل القرار المسند للمرحلة القادمة حيث ذكرى بحقه التي نحققها  
قيمة حسب العلاقة السابقة

ملاحظة: // في قدم القرار  $r$  المسند هي نفس الرقيم للخط الموجوده على المسار ورائف بدلاً  
بالمرحلة (2-2)

$$D[2,2] = 7, D[2,3] = 6$$

$$D[2,4] = 8$$

$$D[2,5] = 8$$

$$D[1,1] = 7$$

وباعتبار المسار الأقل كلفة هو

$$S = 1, V_2, V_1, \dots, V_3, T = 12$$

حيث

$S$  تمثل النقطة الأولى وهي الأقل كلفة

$$7 = V(2)$$

$$7 = V(3)$$

$$10 = V(4)$$

$12 = T$  وتمثل النقطة الأخيرة

$$V_2 = D[1,1] = 7$$

$$V_1 = D[2, D[1,1]] = D[2,7] = 7$$

$$V_3 = D[2, D[1,1]] = D[3,7] = 10$$

ملاحظة: // يجب تحديد النقطة الأولى بغير إلى ذلك المجال التالي ما هي النقطة الأقل التي يجب

أن نختارها في المرحلة التالية

مستخرج من المسار الأفضل هي  $(T) (12, 10, 7, 2, S)$

والآن سوف نقوم بكتابة الخوارزمية لحل هذه المسألة (خوارزمية المخطط متعدد المراحل

المسطرة لتطبيقه الصمعيه )

تتضمن خوارزمية المخطط متعدد المراحل المداخلة الطريقة الصمعيه في العقد  $V$  مرتبه من

1 إلى  $n$  من عقد المسألة  $S$  على التمرس 1 ثم نحسب عقد المجموعة  $V_2$  فهايس متتالية

حتى نصل إلى عقدة التهايه  $T$  أي في التهايس المسطرة بعد المجموعة  $V_2 + 1$  تكون أكبر من

تلك المسطرة  $V[i]$

```

void PGraph (Graph G ,int k ,int n ,int P[])
{ float cost [maxsize]
  int D[maxsize] ,x ,
  cost[n]=0.0
  for (int j= n-1 ; j>= 1 ; j--)
  { // Compute cost[ ]
    Let x be Vertex Suchthat <j,x> is an edge of G and c[j][x]+cost[x]
    is minimum ,
    Cost[j]= c[ ][x]+ cost[x]
    D[j] = x ,
  }
  P[1]=1 , p[k]=n,
  For(j= 2, j<= k-1, j++)
    P[j]=D[ p[j-1]] ;
}

```

- ونلاحظ هنا أننا نحتاج إلى معرفة اسم كل عقدة :
- إن المصفوفة [ P ] تحوي أرقام العقد الموجودة على المسار ، ونلاحظ أن كل عقدة مصفوفة واحدة وتحت كل عقدة رقم مصفوفة [ cost ] لها العقدة x قبل عقدة العقدة المرحلية في المرحلية التالية
- إن أول عقدة نبدأ بحساب التكلفة لها هي العقدة الأخيرة تلك هي  $cost[n]=0.0$
- إذ يجب أن نبدأ بالاعتماد على (كل عقدة x متصلة قبلها بـ عقدة واحدة) يجب أن تكون الآن هي الأولى
- ولابد من مخطط Graph ومن هنا نلاحظ أن C يتم كل شيء
- المخطط يتكون من مصفوفة مؤشرات إلى بيوت (قد نرى طريقة لوضع التجارب) كما يلي

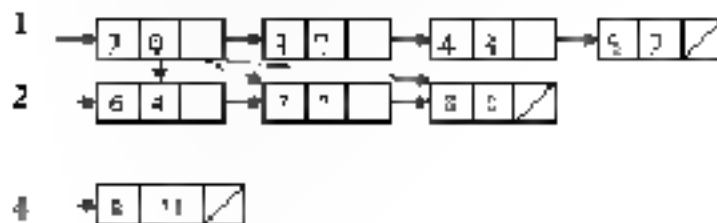
```

Struct node
{
    int vertex;
    float weight;
    Struct node*Link;
}

```

- `vertex` : يمثل بالمعبر ، `weight` : هي القيمة الموجودة على الحافة التي تمثل بالمعبر `weight` ، نضيف هذا الجراء من الجرافيك بمجموعة واحدة فقط
- `Link` : هي المصيرة لمجموعة بعد فتح كائني

```
Struct node*headnodes[n];
```



headnodes



- لاحظ ان المجموعة الاولى ترابط بطريقة مغلقة يعني ان تلك استخدمنا `headnodes`
- ان عملية انشاء وفتح الاليفات ونضيف المخطط يجب ان تتم قبل تنفيذ الخوارزميه `FGraph` ومن ثم يتم استخدامها

وقدنا في جزء البرمجة الخاص بهذه الطريقة التصاعدي (Forward approach)

```

FGraph ( Type head*node.)
{
    Type *p=newType;
    p->Link;
    p->weight;
    p->vertex;
    for(int i=2;i<=m;i++)
    {
        type *p
        p->weight;
    }
}

```

```

        p->vertex;
        q->link=p;
    }
    q->link=0;
}
head[u]=0;

void FGraph (Type *head[26],int k,int n)
{ float cost [100],x, int p[100],D[100]
  cost[n]=0;
  for (int j=n-1;j>=1;j--)
  {cost[j]=head[j]->weight+cost[head[j]->vertex],
    D[j]=head[j]->vertex;
    Head[j]=head[j]->link;
    While (head[j] != 0)
    { x=head[j]->weight +cost[head[j]->vertex]
      If (x<cost[j])
      { cost[j]=x,
        D[j]=head[j]->vertex;
      }
      head[j]=head[j]->link;
    }
  }
}

P[1]=1, p[k]=0
for(int j=2;j<=k-1;j++)
    P[j]=D[p[j-1]];
for(int i=1,i<=k-1,i++)
    cout<< p[i]<< " ";
}

```

### تعبير بعبارة الدالة (FGraph)

في حالة تنفيذ الخوارزمية القائمة على التكرار (Linked List) فلا يمكن إيجادها في وقت يتناسب مع درجة العقدة (j) (مع ارتباطات العقد مع العقد التي بعدها) (الأسهم) حيث (j) هي من 1 إلى n لذا كل الخوارزمية  $O(E)$  أي بحرف قاي وقت بدرجة fox الأولى هو  $O(|V| + E)$  حيث V يمثل عدد العقد و E هي تمثل عدد الحواف

هو بالقيمة فوقه حولة  $f_{\text{opt}}$  النجبه لهر  $\Theta(k)$  حيث  $k$  يمثل عدد المراحل  
هذا يعني ان التكلفة بالقيمة للحواري منه هي

$$\Theta(k) = \sum_{i=1}^k c_i$$

بالإضافة إلى الختر، التي تنطبقه امتدادات "توجد حجة" بوجود خزن التكلفة في المصفوفة  
PD

### 2-6-5: الطريقة التكرارية (Backward approach):

إن تحديد التكلفة هنا يكون من النهاية إلى البداية أي من (T إلى S) حيث هذه الطريقة حلاً  
مباشراً وعلاقة حساب التكلفة مع كالاتي.

$$b_{\text{cost}}(i, j) = \min_{\substack{r \in E \\ r \in V}} \{ b_{\text{cost}}(i-1, r) + c[r, j] \} \quad \text{حيث } i \in E, j \in V$$

يتم كل التكلفة المرحلية في المرحلة التالية  $b_{\text{cost}}(i, j)$  في نفس التكلفة التكلفة التي تربط نقطة  
i إلى نقطة التالية

$$b_{\text{cost}}(i, j) = \begin{cases} \infty & \text{if } i, j \notin E \\ c[i, j] & \text{if } i, j \in E \end{cases}$$

يتم صياغة التكلفة  $b_{\text{cost}}(i, j)$  في نفس التكلفة من النقطة S إلى النقطة (j) هي المصفوفة  $b_{\text{cost}}$   
وتكتب  $b_{\text{cost}}(i, j)$  هي قيمة التكلفة  $b_{\text{cost}}(i, j)$  بوجود العلاقة  $b_{\text{cost}}(i, j) = b_{\text{cost}}(i, j) + c[i, j]$   
حساب  $b_{\text{cost}}(i, j)$  هناك بحسب  $b_{\text{cost}}(i, j)$  بقيمة  $b_{\text{cost}}(i, j) = b_{\text{cost}}(i, j) + c[i, j]$  وهكذا بالتسوية بينه النقطة

المصفوفة السبق المرسومة في الشكل (24) نقول لتكلفة التكلفة في أول مرحلة حسابها هي  
التكلفة من هنا يعني أننا نلاحظ أن التكلفة التكلفة

$$b_{\text{cost}}(3, 6) = \min \{ b_{\text{cost}}(2, 2) + 4, b_{\text{cost}}(2, 3) + 2 \} = 9$$

بصورة للسيطرة على نتيج للخطأ خطأ أسفل التكلفة التي تعطي القيمة لأحد من هنا يعني في  
حساب أو التكلفة التكلفة  
والتكلفة التكلفة بالتسوية التكلفة التكلفة

$$\begin{aligned} b_{\text{cost}}(3, 7) &= 1, \\ b_{\text{cost}}(3, 8) &= 10, \\ b_{\text{cost}}(4, 9) &= 15, \\ b_{\text{cost}}(4, 10) &= 14, \\ b_{\text{cost}}(4, 11) &= 16 \end{aligned}$$



$$D \leq \cos f(5,12) = 16$$

أما بالنسبة لإيجاد أقرب من فلز القصور الأقل كلفة من ( T إلى S ) كان يكلفه مقدار 16  
ولتحديد المسار بسجل القصور العشرة في كل حلقة (تعددة) كما يلي

$$D[3,6] = 3, D[3,7] = 2, D[3,8] = 2$$

$$D[4,9] = 6, D[4,10] = 7, D[4,11] = 8$$

$$D[5,12] = 10$$

$$P[1] = 1$$

$$P[4] = 10$$

$$P[3] = 7$$

$$P[2] = 2$$

$$P[n] = 12$$

وباعتبار المسار الأقل كلفة هو

$$S = 1, V_2, V_3, V_4, T = 12$$

حيث

S يمثل العدد الأولي وفي الإثن دائما

$$2 = V(2)$$

$$7 = V(3)$$

$$10 = V(4)$$

12 = T يمثل العدد الأخير

$$V_2 = D[3, D[4,10]] = D[3,7] = 2$$

$$V_3 = D[4, D[5,12]] = D[4,10] = 7$$

$$V_4 = D[5,12] = 10$$

نستنتج إن المسار الأمثل هو : (T) 12, 10, 7, 2, 1(S)

وهذه هي الخوارزمية الشخصية بالخطوط معتمد على حل المسألة بالطريقة الشخصية  
(BGraph)

```

void BGraph (Graph G ,int k ,int n ,int 2[])
{ float cost [maxsize],
  int D [maxsize] ,r;
  bcost[1]= 0.0;
  for (int i=2; i<=n ,i++)
  {   Compute bcost[i].
      Let r be Vertex Suchthat <r,i> is an edge of G and bcost[r]+c[r][i]
      is minimum;
      bcost[i]= bcost[r] + c[r][i],
      D[i]= r;
  }
  P[1]=1, p[k]=n;
  For(j: k-1, j<=2 j-)
      P[j]=D[ p[j+1] ]
}

```

و هذا هو جزء البرنامج الخامس بهذه الطريقة التفصيلية

```

FGraph (Type head*node )
{ Type *p=newType;
  q=p;
  p->weight;
  p->vertex;
  for(int i=2; i<=m; i++)
  { type *p;
    p->weight;
    p->vertex;
    q->link=p;
  }
  q->link=NULL;
}
head[n]=0;

void BGraph (Type *head[20],int k,int n)
{ float bcost [100],x; int p[100],D[100];
  cost[n]=0;
  for int j=2 j<=n, j++)
  {bcost[j]=head[j] ->weight+bcost[head[j]->vertex];
    D[j]=head[j] ->vertex;
    head[j]=head[j] ->link;
  }
}

```

```

While (head[j] != 0)
{
    x=head[j]->weight + bcost[head[j]->vertex];
    If (x < bcost[j])
        bcost[j]=x;
        D[j]=head[j]->vertex;
    }

    head[j]=head[j]->link;
}

}

P[1]=1; p[x]=n;
for(int i=b; i<=2; i++)
    P[i]=D[p[i-1]];
for(int i=1; i<=k; i++)
    cout<< p[i] << " ";
}

```

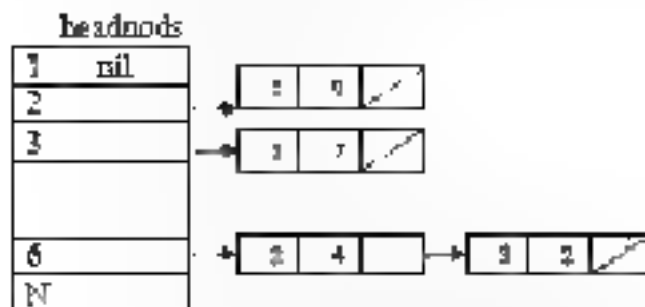
تعيين تعقيداته (EGraph)

في تعقيد الوقت والحرر هذه الخوارزمية هي نفس تلك بتعريف مخزونية (EGraph) ولكن بشرط واحد في تمثيل المخطط ففي الخوارزمية العكسية

هذا يعني في التعقيد الكمية مخزونية هي

$$O = |V| + |E|$$

في خوارزمية الحرر تمثيله تمثيل كالتالي :



في الخوارزمية P فكل نقطة مجهزة بوزن في هذه الحالة يكون الحد  $\leq |V|, |E|$

E فصل الجبر ٤٠ الحرف ٧٢ مثال الحرف في قلعة ايسار

ملحوظة: / هناك تعيينات أخرى للمنظمات المعنية للفراد منها ما يخص الوارد (البيع من العمل يقسم على مجموعته فشرائح)

### 3-6-5 طريقة متتالية الأثر الرجوعا (Back Tracking)

تعتبر إحدى أفكار الطريق للمعرفة القصص من نوع "الرفقاء" حيث يمكن أكثر من حدث للحدث  
(مجموعه من حوال)

الملاحظة / بعد proof نصح لنا القيد في تلك الحيل حيث نبي كيم

بعض. يعظم المبالغ التي بحيث عن مجموعة حلول أو حل أصلي يكون بعض القيود  
يكون. فكل الصيغة  $(x_1, x_2, \dots, x_n)$  حيث  $x_i$  هي مجموعة من قيمته هي  $x_i$   
هذه. رئيسية لهذه الطريقة هي أنه إذا كانت  $x_i$  هي القيمة الجبرية  $x_i$  من  $x_i$  من  $x_i$   
التي حل أصلي في كل محاورات تكون القيود الجبرية هي  $x_i$

تتطلب العديد من المسائل التي يذو حلها باستخدام طريقة اقتداء الاثر رجوعاً (في تحقيق  
الخطوة مجموعة من القيود التي يمكن تقسيمها إلى متين وهناك مجموعة من القيود  
المجموعة التي صريحة (Explicit Constraints) :  
وهي قد نعد لتقدم أولاً الصيغة صيغة هي أن حيث كل المتجهات في نطاق القيود  
الصحيحة تكون قراء الحالات الممكنة

المجموعه 45: ضوابط صناعية (Implicit Constraints)

هي موارد متعدد اي مستويات الصور تحق بناء الحب اي نهائى رقباط <sup>37</sup> يعني هـ

الخطوات المستترة بطريقة التقليل والآثار الجيدة

1. تحديد طرق الحلول الممكنة للتمسك بالإجابة و الحد لتعدد النتائج
2. تقسيم هذا الفرع إلى طرق فرعية بسيطة للبحث (محمي أو مخطط)
3. بحث الفرع بطريقة متعمقة بوزن (Depth First Search) مع استعمال دوال لتقدير (Bounding Function) لتحديد الحركه لم يتم غالب حركته لا نقدر على حق

مثلاً، مسئله صفه n ملکه ( $n$ -queens problem) که ما در فصل قبل دیدیم، به این صورت تعریف می‌شود: برای یک عدد صحیح  $n$ ، یک ماتریس  $n \times n$  را پیدا کنید که هر سطر و هر ستون فقط یک بار دارای یک مقدار غیر صفر باشد.

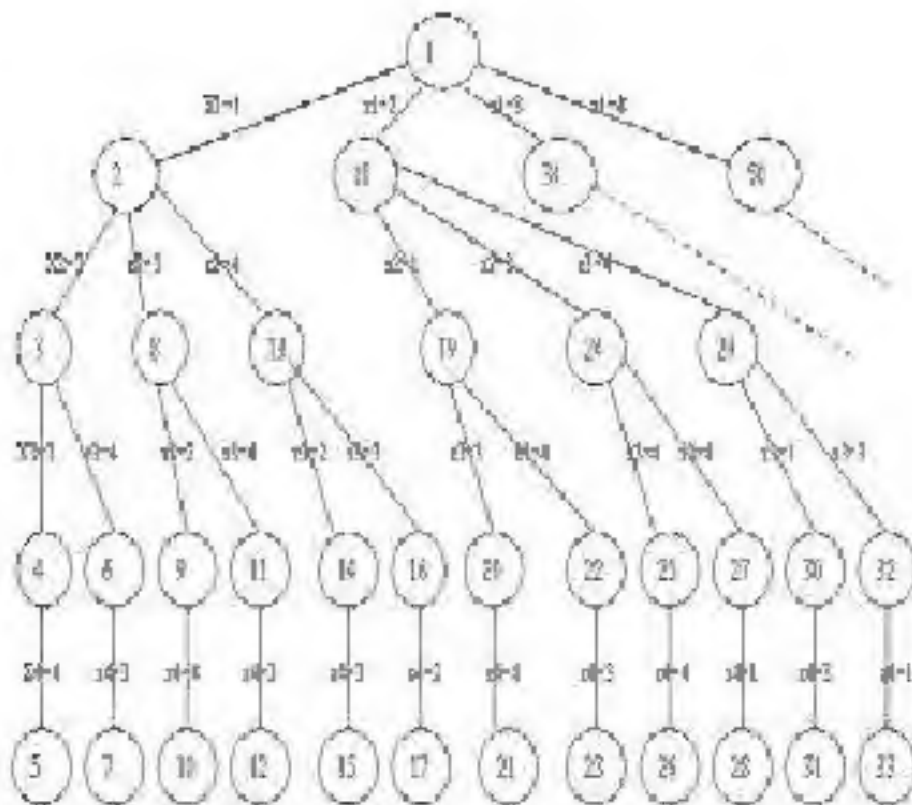
**ملاحظة:** في هذه الأسفل يوجد نيت لإزالة العنكفات بمراد وضعها على لوحة سطونج بعدد ١٠  
بخط لا يوصف أكثر من عشرة على نفس الصفحة ثم المظهر.

لتفرض أن ترتيب صفوف وأعمدة لوحة الشطرنج  $[1 \dots n]$  والمكعب أيضا  $[1 \dots n]$  يمكن وضع المكعب في الصف  $i$  ، عمود  $j$  كل طول عملة  $m$  مكعب بالمتجهات  $X_1, X_2, X_3, \dots, X_m$  حيث  $X_i$  هو العمود الذي توضع عليه المكعب  $i$  بمعنى  $X_i = 2$  فهذا المكعب  $i$  توضع على العمود 2.

القيود الصريحة هنا هي  $S_i = [1 \dots m]$

والقيود الضمنية توصف بارتباطات  $X_i$  بغيرها  
القيود الضمنية هي أن كل المكعب يجب أن تكون على أعمدة وأقطر مختلفة.  
القيود الصريحة تعطى متجهاتها عندما  $m = 3$  ، إن القيد الضمني الأول يشير إلى تشكيل من المتجهات وهذا يفتقر حجم فراغ الحالات أو الحلول من  $m^n$  إلى  $m^3$ .

الآن لدينا 4 صور وعلمنا إيجاد للحلول الممكنة ؟  
العلماء يعرفون أن الحجم هو 4 لذا يمكن رسم شجرة الاحتمالات (التفصيل) كالتالي :



في هذه العملة يجب أن نراعي الشرط الذي يقول بأن العملة يمكن أن توضع في العمود الأول أو الثاني أو الثالث أي هل نضع كجداية الحل لأن المصفوفة فارغة.

بما إن مجموعة الحلول هي 12 هذا يعني أنه لدينا (24) حالة وإن جزء منها يحقق الحل المطلوب .  
 الحراف من قيمة من لكل للقيم الممكنة  $X_i$  ،  
 الحراف من المستوى  $i$  إلى المستوى  $i+1$  تحدد قيم  $X_i$  بلهذا فإن الشجرة على أقصى اليسار تحتوي على كل الحلول ثلاث  $X_i$  تساوي (1).  
 إن المسار المتولد من العقد التالية ( 1, 18, 29, 32, 33 ) يعطي حالة ممكنة يحقق الشرط ، و هو  $(2, 4, 1, 3)$  كما في المصفوفة التالية :

	1	2	3	4
1		X1		
2				X2
3	X3			
4			X4	

هذا  $n=1$  نقصد بها الملكة الأولى وهكذا بالترتيب لبقية الملكات .  
 أي إن:  
 الملكة 1 توضع بالعمود الثاني  
 الملكة 2 توضع بالعمود الرابع  
 الملكة 3 توضع بالعمود الأول  
 الملكة 4 توضع بالعمود الثالث

تمرين // أكتب برنامج يقوم بتطبيق الخوارزمية الخاصة بمسألة (n\_queens problem) ؟

### References:

- 1-Aho, Alfred V. and Jeffrey D. Ullman [1983]. Data Structures and Algorithms. Addison-Wesley, Reading, Massachusetts.
- 2-Bailer J.:An Introduction to Data Structures ; Allyn and Beacon,Inc,1982.
- 3-Berman A.M.: Data Structures via C++ (objects by Evolution ), Oxford University press Inc. ,1997.
- 4-Bertiss A.T. : Data Structures ,theory and practice ; Academic press Inc ,1975.
- 5-Cormen, Thomas H, Charles E. Leiserson and Ronald L. Rivest [1990]. Introduction to Algorithms. McGraw-Hill, New York.
- 6-Dahl O. J. ,Dijkstra E. W and Hoare C.A.R. : Structured 6-programming,Academic press Inc,1972.
- 7-Dale N. and Lilly S.C: pascal plus Data Structures ,Algorithms and Advance programming ,D.C.Heath and company ,1985.
- 8-Goodrich M.T. and Tamassia R. : Data Structures and algorithms in java ;John Wiley and Son Inc. ,1998.
- 9-Gonnet G.H and Beeza -Yates R.:Handbook of Algorithms and data Structures ; Addison\_Wesley,1991.
- 10-Horowitz E.and Sahni S.: Fundamentals of data Structures in pascal ;Computer Science press Inc,1987.
- 11-Knuth, Donald E. [1998]. The Art of Computer Programming, Volume 3, Sorting and Searching. Addison-Wesley, Reading, Massachusetts.
- 12-Pearson, Peter K [1990]. Fast Hashing of Variable-Length Text Strings.
- 13-Communications of the ACM, 33(6):677-680, June 1990.
- 14-Pugh, William [1990]. Skip lists: A Probabilistic Alternative To Balanced Trees.
- 15-Communications of the ACM, 33(6):668-676, June 1990.
- 16-Stephens, Rod [1998]. Ready-to-Run Visual Basic Algorithms. John Wiley & Sons, Inc., New York.
- 17-Thomas H. Cormen (Charles E. Leiserson (Ronald L. Rivest and Clifford Stein. Introduction to Algorithms (Second Edition.
- 18-MIT Press and McGraw-Hill, 2001 ISBN7-03293-262-0 .Chapter 1: Foundations, pp.3-122 .
- 19-C.L. PHILIPS& H.T.NAGLE, Digital Control System: Analysis and Design (Prinice- Hall 1984).

- 20-<http://i136.photobucket.com/albums/q175/uraminza/tab1,2,3.jpg>,
- 21-<http://alyaseer.net/files/file.php?id=10>
- 22-<http://akosai.hajznet.com/bubble-sortalgorithm.pdf>
- 23-<http://akosai.hajznet.com/heap-sortalgorithm.pdf>
- 24-<http://akosai.hajznet.com/merge-sortalgorithm.pdf>